Oracle® AI Database

XML Developer's Kit Programmer's Guide

26ai
G44300-01
October 2025

Oracle AI Database XML Developer's Kit Programmer's Guide, 26ai

G44300-01

Copyright © 1999, 2025, Oracle and/or its affiliates.

Primary Authors: Jiji Thomas, Apoorva Srinivas, Lance Ashdown, Janis Greenberg, Sheila Moore, Sue Pelski

Contributors: Nipun Agarwal, Geeta Arora, Vikas Arora, Thomas Baby, Janet Blowney, Dan Chiba, Steve Ding, Mark Drake, Beda Hammerschmidt, Bill Han, Roza Leyderman, Dmitry Lychagin, Valarie Moore, Steve Muench, Ravi Murthy, Maxim Orgiyan, Mark Scardina, Helen Slattery, Joshua Spiegel, Asha Tarachandani, Jinyu Wang, Simon Wong, Tim Yu, Kongyi Zhou

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle®, Java, MySQL, and NetSuite are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

Preface

Audience i
Related Documents i
Examples i
Conventions ii

Changes in This Release

Desupported Feature i
Deprecated Features i

1 Introduction to Oracle XML Developer's Kit

1.1 Overview of XDK 1
1.2 XDK Components 3
1.2.1 XML Parsers 3
1.2.2 XSLT Processors 4
1.2.3 XML Schema Processors 5
1.2.4 XML Class Generators 5
1.2.5 XML Pipeline Processor 6
1.2.6 Oracle XML SQL Utility 7
1.2.6.1 XML Document Representations 7
1.2.6.2 Using XSU with an XML Class Generator 8
1.2.7 TransX Utility Overview 8
1.2.8 XSQL Pages Publishing Framework 8
1.2.9 SOAP Services 9
1.2.10 XSLT Virtual Machine 10
1.3 Generating XML Documents Using XDK 10
1.3.1 XML Document Generation with Java 10
1.3.2 XML Document Generation with C 12
1.3.3 XML Document Generation with C++ 13
1.4 Development Tools and Frameworks for XDK 14
1.4.1 Oracle JDeveloper 14
1.4.2 Oracle Data Provider for .NET 16

Copyright © 1999, 2025, Oracle and/or its affiliates.

1.5 About Installing XDK 16

2 Security Considerations for Oracle XML Developer's Kit

2.1 Implementing Security for Java 1
2.1.1 Securing XSLT Processing with Oracle XML Developer's Kit 1
2.1.2 Using the Oracle XML Parser Safely 2
2.2 Implementing Security for C 4
2.3 Security for C++ 5

Part I Oracle XML Developer's Kit for C

3 Getting Started with Oracle XML Developer's Kit for C

3.1 Installing XDK for C Components 1
3.2 Configuring the UNIX Environment for XDK for C Components 3
3.2.1 XDK for C Component Dependencies on UNIX 3
3.2.2 Setting Up XDK for C Environment Variables on UNIX 4
3.2.3 Testing the XDK for C Runtime Environment on UNIX 4
3.2.4 Setting Up and Testing the XDK C Compile-Time Environment on UNIX 5
3.2.4.1 Testing the XDK for C Compile-Time Environment on UNIX 5
3.2.5 Verifying the XDK for C Component Version on UNIX 6
3.3 Configuring the Windows Environment for XDK C Components 6
3.3.1 XDK for C Component Dependencies on Windows 6
3.3.2 Setting Up XDK for C Environment Variables on Windows 7
3.3.3 Testing the XDK for C Runtime Environment on Windows 7
3.3.4 Setting Up and Testing the XDK for C Compile-Time Environment on Windows 8
3.3.4.1 Testing the XDK for C Compile-Time Environment on Windows 8
3.3.5 Using the XDK for C Components and Visual C++ in Microsoft Visual Studio 9
3.3.5.1 Setting a Path for a Project in Visual C++ on Windows 9
3.3.5.2 Setting the Library Path in Visual C++ on Windows 10
3.4 Overview of the Unified C API 11
3.5 Globalization Support for the XDK for C Components 12

4 Using the XSLT and XVM Processors for C

4.1 XSLT XVM Processor 1
4.1.1 XVM Usage Example 1
4.1.2 Using the XVM Processor Command-Line Utility 3
4.1.3 Accessing the XVM Processor for C 3
4.2 XSLT Processor for XDK for C 3
4.2.1 XSLT Processor Usage Example 3

Copyright © 1999, 2025, Oracle and/or its affiliates.

4.2.2 XPath Processor Usage Example 4
4.2.3 Using the C XSLT Processor Command-Line Utility 5
4.2.4 Accessing Oracle XSLT processor for C 6
4.3 Using the Demo Files Included with the Software 6
4.3.1 Building the C Demo Programs for XSLT 6

5 Using the XML Parser for C

5.1 Introduction to the XML Parser for C 1
5.1.1 Prerequisites for Using the XML Parser for C 1
5.1.2 Standards and Specifications for the XML Parser for C 1
5.2 Using the XML Parser API for C 2
5.2.1 Overview of the Parser API for C 2
5.2.1.1 XML Parser for C Data Types 3
5.2.1.2 XML Parser for C Defaults 3
5.2.2 XML Parser for C Calling Sequence 4
5.2.3 Using the XML Parser for C: Basic Process 6
5.2.4 Running the XML Parser for C Demo Programs 8
5.2.5 Using the C XML Parser Command-Line Utility 9
5.2.5.1 Using the XML Parser Command-Line Utility: Example 10
5.3 Using the DOM API for C 11
5.3.1 Controlling the Data Encoding of XML Documents for the C API 11
5.3.2 Using NULL-Terminated and Length-Encoded C API Functions 12
5.3.3 Handling Errors with the C API 12
5.4 Using orastream Functions 13
5.5 Using the SAX API for C 16
5.6 Using the XML Pull Parser for C 16
5.6.1 Using Basic XML Pull Parsing Capabilities 16
5.6.1.1 XML Event Context 17
5.6.1.2 About the XML Event Context 17
5.6.2 Parsing Multiple XML Documents 17
5.6.3 ID Callback 18
5.6.4 Error Handling for the XML Pull Parser 18
5.6.4.1 Parser Errors 18
5.6.4.2 Programming Errors 19
5.6.5 Sample Pull Parser Application 19
5.7 Using OCI and the XDK for C API 21
5.7.1 Using XMLType Functions and Descriptions 21
5.7.2 Initializing an XML Context for Oracle XML DB 21
5.7.3 Creating XMLType Instances on the Client 22
5.7.4 Operating on XML Data in the Database Server 23

Copyright © 1999, 2025, Oracle and/or its affiliates.

5.7.5 Using OCI and the XDK for C API: Examples 23

6 Using Binary XML with C

6.1 Introduction to Binary XML for C 1
6.2 Prerequisites for Using Binary XML with C 1
6.3 Binary XML Storage Format – C 1

7 Using the XML Schema Processor for C

7.1 Oracle XML Schema Processor for C 1
7.1.1 Oracle XML Schema for C Features 1
7.1.2 Standards Conformance for Oracle XML Schema Processor for C 2
7.1.3 XML Schema Processor for C: Supplied Software 2
7.2 Using the C XML Schema Processor Command-Line Utility 2
7.3 XML Schema Processor for C Usage Diagram 3
7.4 How to Run XML Schema for C Sample Programs 4
7.5 What Is the Streaming Validator? 5
7.5.1 Using Transparent Mode 5
7.5.1.1 Error Handling in Transparent Mode 5
7.5.1.2 Streaming Validator Example 6
7.5.2 Using Opaque Mode 7
7.5.2.1 Error Handling in Opaque Mode 7
7.5.2.2 Example of Opaque Mode Application 7
7.5.3 Using Function XmlSchemaLoad() With an Existing DOM 8
7.5.4 Validation Options 8

8 Determining XML Differences Using C

8.1 Overview of XMLDiff in C 1
8.1.1 Process Flow for XMLDiff 1
8.2 Using XmlDiff 1
8.2.1 User Options for Comparison Optimization 1
8.2.2 User Option for Hashing 2
8.2.3 How XmlDiff Looks at Input Documents 2
8.2.4 Using the XmlDiff Command-Line Utility 2
8.2.5 Sample Input Document 3
8.2.6 Sample Xdiff Instance Document 4
8.2.7 Output Model and XML Processing Instructions 5
8.2.8 Xdiff Operations 5
8.2.9 Format of Xdiff Instance Document 6
8.2.10 Xdiff Schema 6

Copyright © 1999, 2025, Oracle and/or its affiliates.

8.2.11 Using XMLDiff in an Application 9
8.2.12 Customized Output 10
8.3 Using XmlPatch 11
8.3.1 Using the XmlPatch Command-Line Utility 12
8.3.2 Using XmlPatch in an Application 12
8.4 Using XmlHash 13
8.4.1 Invoking XmlDiff and XmlPatch 14

9 Using SOAP with the Oracle XML Developer's Kit for C

9.1 Introduction to SOAP for C 1
9.1.1 SOAP Messaging Overview 1
9.1.1.1 SOAP Message Format 2
9.1.2 Using SOAP Clients 3
9.1.3 Using SOAP Servers 4
9.2 SOAP C Functions 4
9.3 SOAP Example 1: Sending an XML Document 6
9.4 SOAP Example 2: A Response Asking for Clarification 12
9.5 SOAP Example 3: Using POST 14

Part II Oracle XML Developer's Kit for Java

10 Unified Java API for XML

10.1 Overview of Unified Java API for XML 1
10.2 Component Unification 1
10.3 About Moving to the Unified Java API 1
10.3.1 Java DOM APIs for XMLType Classes 2
10.3.2 Extension APIs 3
10.3.3 Document Creation Java APIs 3

11 Getting Started with Oracle XML Developer's Kit for Java

11.1 Installing XDK for Java Components 1
11.2 XDK for Java Component Dependencies 2
11.3 Setting Up the XDK for Java Environment 5
11.3.1 Setting Up XDK for Java Environment Variables for UNIX 5
11.3.2 Testing the XDK for Java Environment on UNIX 6
11.3.3 Setting Up XDK for Java Environment Variables for Windows 7
11.3.4 Testing the XDK for Java Environment on Windows 8
11.4 Verifying the XDK (Java) Version 9

Copyright © 1999, 2025, Oracle and/or its affiliates.

12 XML Parsing for Java

12.1 Introduction to XML Parsing for Java 1
12.1.1 Prerequisites for Parsing with Java 1
12.1.2 Standards and Specifications for XML Parsing for Java 1
12.1.3 Large Node Handling 2
12.1.4 XML Parsing in Java: Overview 2
12.1.5 DOM in XML Parsing 4
12.1.5.1 DOM Creation 4
12.1.6 SDOM 4
12.1.6.1 Pluggable DOM Support 5
12.1.6.2 Lazy Materialization 5
12.1.6.3 Configurable DOM Settings 5
12.1.6.4 DOM Support for Fast Infoset 5
12.1.7 SAX in the XML Parser 6
12.1.8 JAXP in the XML Parser 7
12.1.9 Namespace Support in the XML Parser 7
12.1.10 Validation in the XML Parser 8
12.1.11 Compression in the XML Parser 9
12.2 Using XML Parsing for Java: Overview 11
12.2.1 Using the XML Parser for Java: Basic Process 11
12.2.2 Running the XML Parser for Java Demo Programs 12
12.2.3 Using the Java XML Parser Command-Line Utility (oraxml) 14
12.3 Parsing XML with DOM 14
12.3.1 Using the DOM API for Java 15
12.3.2 DOM Parser Architecture 15
12.3.3 Performing Basic DOM Parsing 16
12.3.4 Creating SDOM 20
12.3.4.1 Using SDOM 20
12.3.4.2 Using Lazy Materialization 21
12.3.4.3 Using Configurable DOM Settings 24
12.3.4.4 Using Fast Infoset with SDOM 25
12.3.4.5 SDOM Applications 26
12.3.4.6 XDK Java DOM Improvements 26
12.3.5 Performing DOM Operations with Namespaces 27
12.3.6 Performing DOM Operations with Events 28
12.3.7 Performing DOM Operations with Ranges 29
12.3.8 Performing DOM Operations with TreeWalker 31
12.4 Parsing XML with SAX 33
12.4.1 Using the SAX API for Java 33
12.4.2 Performing Basic SAX Parsing 35
12.4.3 Performing Basic SAX Parsing with Namespaces 37

Copyright © 1999, 2025, Oracle and/or its affiliates.

12.4.4 Performing SAX Parsing with XMLTokenizer 38
12.5 Parsing XML with JAXP 40
12.5.1 JAXP Structure 40
12.5.2 Using the SAX API Through JAXP 40
12.5.3 Using the DOM API Through JAXP 41
12.5.4 Transforming XML Through JAXP 41
12.5.5 Parsing with JAXP 42
12.5.6 Performing Basic Transformations with JAXP 44
12.6 Compressing and Decompressing XML 44
12.6.1 Compressing a DOM Object 45
12.6.2 Decompressing a DOM Object 45
12.6.3 Compressing a SAX Object 46
12.6.4 Decompressing a SAX Object 47
12.7 Tips and Techniques for Parsing XML 47
12.7.1 Extracting Node Values from a DOM Tree 47
12.7.2 Merging Documents with appendChild() 48
12.7.3 Parsing DTDs 49
12.7.3.1 Loading External DTDs 49
12.7.3.2 Caching DTDs with setDoctype 50
12.7.4 Handling Character Sets with the XML Parser 52
12.7.4.1 Detecting the Encoding of an XML File on the Operating System 52
12.7.4.2 Preventing Distortion of XML Stored in an NCLOB Column 52
12.7.4.3 Writing an XML File in a Nondefault Encoding 53
12.7.4.4 Parsing XML Stored in Strings 53
12.7.4.5 Parsing XML Documents with Accented Characters 54
12.7.4.6 Handling Special Characters in Tag Names 55

13 Using Binary XML with Java

13.1 Introduction to Binary XML for Java 1
13.1.1 Binary XML Storage Format – Java 1
13.1.2 Binary XML Processors 1
13.2 Models for Using Binary XML 2
13.2.1 Usage Terminology for Binary XML 2
13.2.2 Standalone Model 2
13.2.3 Client/Server Model 2
13.2.4 Web Services Model With Repository 3
13.2.5 Web Services Model Without Repository 3
13.3 Components of Binary XML for Java 3
13.3.1 Binary XML Encoding 4
13.3.2 Binary XML Decoding 5
13.4 Binary XML Vocabulary Management 5

Copyright © 1999, 2025, Oracle and/or its affiliates.

13.4.1 Schema Management 5
13.4.1.1 Schema Registration for Binary XML Vocabulary Management 5
13.4.1.2 Schema Identification 6
13.4.1.3 Schema Annotations 6
13.4.1.4 User-Level Annotations 6
13.4.1.5 System-Level Annotations 6
13.4.2 Token Management 6
13.5 Using the Java Binary XML Package 6
13.5.1 Binary XML Encoder 7
13.5.1.1 Schema-Less Option 7
13.5.1.2 Inline-Token Option 8
13.5.2 Binary XML Decoder 8
13.5.3 Schema Registration Overview 9
13.5.4 Resolving xsi:schemaLocation 9
13.5.5 Binary XML 9
13.5.6 Persistent Storage of Metadata 10

14 Using the XSLT Processor for Java

14.1 Introduction to the XSLT Processor 1
14.1.1 Prerequisites for Using the XSLT Processor for Java 1
14.1.2 Standards and Specifications for the XSLT Processor for Java 1
14.1.3 XML Transformation with XSLT 1.0 and 2.0 2
14.2 Using the XSLT Processor for Java: Overview 3
14.2.1 Using the XSLT Processor for Java: Basic Process 3
14.2.2 Running the XSLT Processor Demo Programs 4
14.2.3 Using the XSLT Processor Command-Line Utility 6
14.2.3.1 Using the XSLT Processor Command-Line Utility: Example 7
14.3 Transforming XML 8
14.3.1 Performing Basic XSL Transformation 8
14.3.2 Getting DOM Results from an XSL Transformation 10
14.4 Programming with Oracle XSLT Extensions 10
14.4.1 Overview of Oracle XSLT Extensions 10
14.4.2 Specifying Namespaces for XSLT Extension Functions 11
14.4.3 Using Static and Nonstatic Java Methods in XSLT 11
14.4.4 Using Constructor Extension Functions 12
14.4.5 Using Return Value Extension Functions 12
14.5 Tips and Techniques for Transforming XML 14
14.5.1 Merging XML Documents with XSLT 14
14.5.2 Creating an HTML Input Form Based on the Columns in a Table 15

Copyright © 1999, 2025, Oracle and/or its affiliates.

15 Using the XQuery Processor for Java

15.1 Introduction to the XQuery Processor for Java 1
15.2 XQJ Entity Resolution 2
15.2.1 Resolution of Documents for fn:doc 2
15.2.2 Resolution of External XQuery Functions 4
15.2.3 Resolution of Imported XQuery Modules 7
15.2.4 Resolution of XML Schemas Imported by an XQuery Query 9
15.2.5 Prefabricated Entity Resolvers for XQuery 10
15.2.6 Resolution of Other Types of Entity 12
15.3 XQuery Output Declarations 12
15.4 Improving Application Performance and Scalability with XQuery 14
15.4.1 Streaming Query Evaluation 15
15.4.2 External Storage 16
15.4.3 Thread Safety for XQJ 18
15.5 Performing Updates 18
15.6 Oracle XQuery Functions and Operators 20
15.6.1 Oracle XQuery Functions for Duration, Date, and Time 20
15.6.1.1 ora-fn:date-from-string-with-format 21
15.6.1.2 ora-fn:date-to-string-with-format 21
15.6.1.3 ora-fn:dateTime-from-string-with-format 22
15.6.1.4 ora-fn:dateTime-to-string-with-format 22
15.6.1.5 ora-fn:time-from-string-with-format 23
15.6.1.6 ora-fn:time-to-string-with-format 24
15.6.1.7 Format Argument 24
15.6.1.8 Locale Argument 24
15.6.2 Oracle XQuery Functions for Strings 25
15.6.2.1 ora-fn:pad-left 25
15.6.2.2 ora-fn:pad-right 26
15.6.2.3 ora-fn:trim 27
15.6.2.4 ora-fn:trim-left 27
15.6.2.5 ora-fn:trim-right 27
15.7 Standards and Specifications for the XQuery Processor for Java 28
15.7.1 Optional XQuery Features 28
15.7.2 Implementation-Defined Items 29

16 Using XQuery API for Java to Access Oracle XML DB

16.1 Introduction to Oracle XML DB Support for XQJ 1
16.1.1 Prerequisites for Using XQJ to Access Oracle XML DB 1
16.2 Examples: Using XQJ to Query Oracle XML DB 2
16.3 XQJ Support for Oracle XML DB 5

Copyright © 1999, 2025, Oracle and/or its affiliates.

16.3.1 Other Oracle XML DB XQJ Support Limitations 6
16.4 XQJ Performance Considerations for Use with Oracle XML DB 7

17 Using the XML Schema Processor for Java

17.1 Introduction to XML Validation 1
17.1.1 Prerequisites for Using the XML Schema Processor for Java 1
17.1.2 Standards and Specifications for the XML Schema Processor for Java 1
17.1.3 XML Validation with DTDs 1
17.1.3.1 DTD Samples in XDK 2
17.1.4 XML Validation with XML Schemas 3
17.1.4.1 XML Schema Samples in XDK 3
17.1.5 Differences Between XML Schemas and DTDs 5
17.2 Using the XML Schema Processor: Overview 6
17.2.1 Using the XML Schema Processor for Java: Basic Process 7
17.2.2 Running the XML Schema Processor Demo Programs 9
17.2.3 Using the XML Schema Processor Command-Line Utility 11
17.2.3.1 Using oraxml to Validate Against a Schema 11
17.2.3.2 Using oraxml to Validate Against a DTD 12
17.3 Validating XML with XML Schemas 12
17.3.1 Validating Against Internally Referenced XML Schemas 12
17.3.2 Validating Against Externally Referenced XML Schemas 13
17.3.3 Validating a Subsection of an XML Document 15
17.3.4 Validating XML from a SAX Stream 15
17.3.5 Validating XML from a DOM 16
17.3.6 Validating XML from Designed Types and Elements 17
17.4 Tips and Techniques for Programming with XML Schemas 20
17.4.1 Overriding the Schema Location with an Entity Resolver 20
17.4.2 Converting DTDs to XML Schemas 21

18 Using the JAXB Class Generator

18.1 Introduction to the JAXB Class Generator 1
18.1.1 Prerequisites for Using the JAXB Class Generator 1
18.1.2 Standards and Specifications for the JAXB Class Generator 1
18.1.3 JAXB Class Generator Features 2
18.1.4 Marshalling and Unmarshalling with JAXB 2
18.1.5 Validation with JAXB 3
18.1.6 JAXB Customization 3
18.2 Using the JAXB Class Generator: Overview 4
18.2.1 Using the JAXB Processor: Basic Process 4
18.2.2 Running the XML Schema Processor Demo Programs 6

Copyright © 1999, 2025, Oracle and/or its affiliates.

18.2.3 Using the JAXB Class Generator Command-Line Utility 8
18.2.3.1 Using the JAXB Class Generator Command-Line Utility: Example 8
18.2.4 JAXB Features Not Supported in XDK 9
18.3 Processing XML with the JAXB Class Generator 9
18.3.1 Binding Complex Types 9
18.3.1.1 Defining the Schema to Validate sample3.xml 9
18.3.1.2 Generating and Compiling the Java Classes 11
18.3.1.3 Processing the XML Data in sample3.xml 12
18.3.2 Customizing a Class Name in a Top-Level Element 13
18.3.2.1 Defining the Schema to Validate schema10.xml 13
18.3.2.2 Generating and Compiling the Java Classes 15
18.3.2.3 Processing the XML Data in sample10.xml 16

19 Using the XML Pipeline Processor for Java

19.1 Introduction to the XML Pipeline Processor 1
19.1.1 Prerequisites for Using the XML Pipeline Processor for Java 1
19.1.2 Standards and Specifications for the XML Pipeline Processor for Java 1
19.1.3 Multistage XML Processing 2
19.1.4 Customized Pipeline Processes 3
19.2 Using the XML Pipeline Processor for Java: Overview 3
19.2.1 Using the XML Pipeline Processor for Java: Basic Process 4
19.2.2 Running the XML Pipeline Processor Demo Programs 7
19.2.3 Using the XML Pipeline Processor Command-Line Utility 8
19.3 Processing XML in a Pipeline 9
19.3.1 Creating a Pipeline Document 9
19.3.1.1 Example of a Pipeline Document 9
19.3.2 Writing a Pipeline Processor Application 11
19.3.3 Writing a Pipeline Error Handler 12

20 Determining XML Differences Using Java

20.1 Overview of XML Diffing Utilities for Java 1
20.2 User Options for the Java XML Diffing Library 2
20.3 Using Java XML Diffing Methods to Find Differences 3
20.3.1 About the append-node Operation 4
20.3.2 About the insert-node-before Operation 5
20.3.3 About the delete-node Operation 6
20.4 Invoking diff and difftoDoc Methods in a Java Application 6
20.5 Using Java XML hash and equal Methods to Identify and Compare Inputs 10
20.6 Diff Output Schema 10

Copyright © 1999, 2025, Oracle and/or its affiliates.

21 Using the XML SQL Utility

21.1 Introduction to the XML SQL Utility (XSU) 1
21.1.1 Prerequisites for Using the XML SQL Utility (XSU) 1
21.1.2 XSU Features 1
21.1.3 XSU Restrictions 2
21.2 Using the XML SQL Utility: Overview 2
21.2.1 Using XSU: Basic Process 2
21.2.1.1 Generating XML with the XSU Java API: Basic Process 2
21.2.1.2 Performing DML with the XSU Java API: Basic Process 4
21.2.2 Installing XSU 5
21.2.2.1 XSU in the Database 6
21.2.2.2 XSU in an Application Server 6
21.2.2.3 XSU in a Web Server 7
21.2.3 Running the XSU Demo Programs 8
21.2.4 Using the XSU Command-Line Utility 10
21.2.4.1 Generating XML with the XSU Command-Line Utility 12
21.2.4.2 Generating XMLType Data with the XSU Command-Line Utility 12
21.2.4.3 Performing DML with the XSU Command-Line Utility 13
21.3 Programming with the XSU Java API 13
21.3.1 Generating a String with OracleXMLQuery 13
21.3.1.1 Running the testXMLSQL Program 14
21.3.2 Generating a DOM Tree with OracleXMLQuery 15
21.3.3 Paginating Results with OracleXMLQuery 15
21.3.3.1 Limiting the Number of Rows in the Result Set 15
21.3.3.2 Keeping an Object Open for the Duration of the User's Session 16
21.3.3.3 Paginating Results with OracleXMLQuery: Example 16
21.3.4 Generating Scrollable Result Sets 17
21.3.5 Generating XML from Cursor Objects 18
21.3.6 Inserting Rows with OracleXMLSave 19
21.3.6.1 Inserting XML into All Columns with OracleXMLSave 19
21.3.6.2 Inserting XML into a Subset of Columns with OracleXMLSave 20
21.3.7 Updating Rows Using OracleXMLSave 21
21.3.7.1 Updating Key Columns Using OracleXMLSave 21
21.3.7.2 Updating a Column List Using OracleXMLSave 22
21.3.8 Deleting Rows using XSU 23
21.3.8.1 Deleting by Row with OracleXMLSave 24
21.3.8.2 Deleting by Key with OracleXMLSave 24
21.3.9 Handling XSU Java Exceptions 25
21.3.9.1 Getting the Parent Exception 26
21.3.9.2 Raising a No Rows Exception 26
21.4 Tips and Techniques for Programming with XSU 27

Copyright © 1999, 2025, Oracle and/or its affiliates.

21.4.1 How XSU Maps Between SQL and XML 27
21.4.1.1 Default SQL-to-XML Mapping 27
21.4.1.2 Default XML-to-SQL Mapping 29
21.4.1.3 Customizing Generated XML 30
21.4.2 How XSU Processes SQL Statements 32
21.4.2.1 How XSU Queries the Database 32
21.4.2.2 How XSU Inserts Rows 32
21.4.2.3 How XSU Updates Rows 33
21.4.2.4 How XSU Deletes Rows 33
21.4.2.5 How XSU Commits After DML 34

22 Using the TransX Utility

22.1 Introduction to the TransX Utility 1
22.1.1 Prerequisites for Using the TransX Utility 1
22.1.2 TransX Utility Features 2
22.1.2.1 Simplified Multilingual Data Loading 2
22.1.2.2 Simplified Data Format Support and Interface 2
22.1.2.3 Additional TransX Utility Features 2
22.2 Using the TransX Utility: Overview 3
22.2.1 Using the TransX Utility: Basic Process 3
22.2.2 Running the TransX Utility Demo Programs 5
22.2.3 Using the TransX Command-Line Utility 7
22.2.3.1 TransX Utility Command-Line Options 7
22.2.3.2 TransX Utility Command-Line Parameters 8
22.3 Loading Data with the TransX Utility 9
22.3.1 Storing Messages in the Database 9
22.3.2 Creation of a Data Set in a Predefined Format 10
22.3.2.1 Format of the Input XML Document 11
22.3.2.2 Specifying Translations in a Data Set 13
22.3.3 Loading the Data 15
22.3.4 Querying the Data 16

23 Data Loading Format (DLF) Specification

23.1 Introduction to DLF 1
23.1.1 Naming Conventions for DLF 1
23.1.1.1 Elements and Attributes 1
23.1.1.2 Values 1
23.1.1.3 File Extensions 2
23.2 General Structure of DLF 2
23.2.1 Tree Structure of DLF 2

Copyright © 1999, 2025, Oracle and/or its affiliates.

23.3 DLF Specifications 4
23.3.1 XML Declaration in DLF 4
23.3.2 Entity References in DLF 5
23.3.3 Elements in DLF 5
23.3.3.1 Top-Level Table Element 6
23.3.3.2 Translation Elements 6
23.3.3.3 Lookup Key Elements 6
23.3.3.4 Metadata Elements 7
23.3.3.5 Data Elements 7
23.3.4 Attributes in DLF 8
23.3.4.1 DLF Attributes 8
23.3.4.2 XML Namespace Attributes 11
23.4 DLF Examples 12
23.4.1 Minimal DLF Document 12
23.4.2 Typical DLF Document 12
23.4.3 Localized DLF Document 14

24 Using the XSQL Pages Publishing Framework

24.1 Introduction to the XSQL Pages Publishing Framework 1
24.1.1 Prerequisites for Using the XSQL Pages Publishing Framework 2
24.2 Using the XSQL Pages Publishing Framework: Overview 2
24.2.1 Using the XSQL Pages Framework: Basic Process 2
24.2.2 Setting Up the XSQL Pages Framework 5
24.2.2.1 Creating and Testing XSQL Pages with Oracle JDeveloper 5
24.2.2.2 Setting the CLASSPATH for XSQL Pages 6
24.2.2.3 Configuring the XSQL Servlet Container 6
24.2.2.4 Setting Up the Connection Definitions 7
24.2.3 Running the XSQL Pages Demo Programs 8
24.2.3.1 Setting Up the XSQL Demos 10
24.2.3.2 Running the XSQL Demos 10
24.2.4 Using the XSQL Pages Command-Line Utility 11
24.3 Generating and Transforming XML with XSQL Servlet 12
24.3.1 Composing XSQL Pages 12
24.3.1.1 Using Bind Parameters 13
24.3.1.2 Using Lexical Substitution Parameters 15
24.3.1.3 Providing Default Values for Bind and Substitution Parameters 16
24.3.1.4 How the XSQL Page Processor Handles Different Types of Parameters 17
24.3.2 Producing Datagrams from SQL Queries 18
24.3.3 Transforming XML Datagrams into an Alternative XML Format 19
24.3.4 Transforming XML Datagrams into HTML for Display 22
24.4 Using XSQL in Java Programs 23

Copyright © 1999, 2025, Oracle and/or its affiliates.

24.5 XSQL Pages Tips and Techniques 25
24.5.1 XSQL Pages Limitations 25
24.5.2 Hints for Using the XSQL Servlet 25
24.5.2.1 Specifying a DTD While Transforming XSQL Output to a WML Document 25
24.5.2.2 Testing Conditions in XSQL Pages 25
24.5.2.3 Passing a Query Result to the WHERE Clause of Another Query 26
24.5.2.4 Handling Multivalued HTML Form Parameters 26
24.5.2.5 Invoking PL/SQL Wrapper Procedures to Generate XML Datagrams 27
24.5.2.6 Accessing Contents of Posted XML 28
24.5.2.7 Changing Database Connections Dynamically 29
24.5.2.8 Retrieving the Name of the Current XSQL Page 29
24.5.3 Resolving Common XSQL Connection Errors 29
24.5.3.1 Receiving "Unable to Connect" Errors 29
24.5.3.2 Receiving "No Posted Document to Process" When Using HTTP POST 30
24.5.4 Security Considerations for XSQL Pages 30
24.5.4.1 Installing Your XSQL Configuration File in a Safe Directory 30
24.5.4.2 Disabling Default Client Stylesheet Overrides 30
24.5.4.3 Protecting Against the Misuse of Substitution Parameters 31

25 Using the XSQL Pages Publishing Framework: Advanced Topics

25.1 Customizing the XSQL Configuration File Name 1
25.2 Controlling How Stylesheets Are Processed 2
25.2.1 Overriding Client Stylesheets 2
25.2.2 Controlling the Content Type of the Returned Document 2
25.2.3 Assigning the Stylesheet Dynamically 3
25.2.4 Processing XSLT Stylesheets in the Client 4
25.2.5 Providing Multiple Stylesheets 4
25.3 Working with Array-Valued Parameters 5
25.3.1 Supplying Values for Array-Valued Parameters 6
25.3.2 Setting Array-Valued Page or Session Parameters from Strings 7
25.3.3 Binding Array-Valued Parameters in SQL and PL/SQL Statements 8
25.4 Setting Error Parameters on Built-In Actions 10
25.4.1 Using Conditional Logic with Error Parameters 10
25.4.2 Formatting XSQL Action Handler Errors 11
25.5 Including XMLType Query Results in XSQL Pages 12
25.6 Handling Posted XML Content 14
25.6.1 Understanding XML Posting Options 14
25.7 Producing PDF Output with the FOP Serializer 16
25.8 Performing XSQL Customizations 17
25.8.1 Writing Custom XSQL Action Handlers 17
25.8.1.1 Implementing the XSQLActionHandler Interface 18

Copyright © 1999, 2025, Oracle and/or its affiliates.

25.8.1.2 Using Multivalued Parameters in Custom XSQL Actions 21
25.8.2 Implementing Custom XSQL Serializers 21
25.8.2.1 Techniques for Using a Custom Serializer 22
25.8.2.2 Assigning a Short Name to a Custom Serializer 22
25.8.3 Using a Custom XSQL Connection Manager for JDBC Data Sources 24
25.8.4 Writing Custom XSQL Connection Managers 24
25.8.4.1 Accessing Authentication Information in a Custom Connection Manager 25
25.8.5 Implementing a Custom XSQLErrorHandler 26
25.8.6 Providing a Custom XSQL Logger Implementation 26

Part III Oracle XML Developer's Kit for C++

26 Getting Started with Oracle XML Developer's Kit for C++

26.1 Installing XDK for C++ Components 1
26.2 Configuring the UNIX Environment for XDK for C++ Components 1
26.2.1 XDK for C++ Component Dependencies on UNIX 1
26.2.2 Setting Up XDK for C++ Environment Variables on UNIX 2
26.2.3 Testing the XDK for C++ Runtime Environment on UNIX 2
26.2.4 Setting Up and Testing the XDK for C++ Compile-Time Environment on UNIX 2
26.2.4.1 Testing the XDK for C++ Compile-Time Environment on UNIX 2
26.2.5 Verifying the XDK for C++ Component Version on UNIX 3
26.3 Configuring the Windows Environment for XDK for C++ Components 3
26.3.1 XDK for C++ Component Dependencies on Windows 3
26.3.2 Setting Up XDK for C++ Environment Variables on Windows 3
26.3.3 Testing the XDK for C++ Runtime Environment on Windows 3
26.3.4 Setting Up and Testing the XDK for C++ Compile-Time Environment on Windows 3
26.3.4.1 Testing the XDK for C++ Compile-Time Environment on Windows 4
26.3.5 Using the XDK for C++ Components with Visual C/C++ 4

27 Overview of the Unified C++ Interfaces

27.1 What Is the Unified C++ API? 1
27.2 Accessing the C++ Interface 1
27.3 OracleXML Namespace 1
27.3.1 OracleXML Interfaces 2
27.4 Ctx Namespace 2
27.4.1 OracleXML Data Types 2
27.4.2 Ctx Interfaces 2
27.5 IO Namespace 3
27.5.1 IO Data Types 3

Copyright © 1999, 2025, Oracle and/or its affiliates.

27.5.2 IO Interfaces 3
27.6 Tools Package 3
27.6.1 Tools Interfaces 3
27.7 Error Message Files 4

28 Using the XML Parser for C++

28.1 Introduction to Oracle XML Parser for C++ 1
28.2 DOM Namespace 1
28.2.1 DOM Data Types 2
28.2.2 DOM Interfaces 2
28.2.3 DOM Traversal and Range Data Types 3
28.2.4 DOM Traversal and Range Interfaces 3
28.3 Parser Namespace 3
28.3.1 GParser Interface 3
28.3.2 DOMParser Interface 4
28.3.3 SAXParser Interface 4
28.3.3.1 SAX Event Handlers 4
28.4 Thread Safety for the XML Parser for C++ 4
28.5 XML Parser for C++ Usage 4
28.6 XML Parser for C++ Default Behavior 4
28.7 C++ Sample Files 5

29 Using the XSLT Processor for C++

29.1 Accessing XSLT for C++ 1
29.2 XSL Namespace 1
29.2.1 XSL Interfaces 1
29.3 XSLT for C++ DOM Interface Usage 2
29.4 Invoking XSLT for C++ 2
29.4.1 Command-Line Usage 2
29.4.2 Writing C++ Code to Use Supplied APIs 2
29.5 Using the Sample Files Included with the Software 3

30 Using the XML Schema Processor for C++

30.1 Oracle XML Schema Processor for C++ 1
30.1.1 Oracle XML Schema for C++ Features 1
30.1.1.1 Online Documentation 1
30.1.2 Standards Conformance for Oracle XML Schema Processor for C++ 2
30.2 XML Schema Processor API 2
30.2.1 Invoking XML Schema Processor for C++ 2

Copyright © 1999, 2025, Oracle and/or its affiliates.

30.3 Running the Provided XML Schema for C++ Sample Programs 3

31 Using the XPath Processor for C++

31.1 XPath Interfaces 1
31.2 Sample Programs 1

32 Using the XML Class Generator for C++

32.1 Accessing the XML C++ Class Generator 1
32.2 Using the XML C++ Class Generator 1
32.2.1 External DTD Parsing 1
32.3 Using the XML C++ Class Generator Command-Line Utility 1
32.3.1 Input to the XML C++ Class Generator 2
32.4 Using the XML C++ Class Generator Examples 2
32.4.1 XML C++ Class Generator Example 1: XML — Input File to Class Generator, CG.xml 3
32.4.2 XML C++ Class Generator Example 2: DTD — Input File to Class Generator, CG.dtd 3
32.4.3 XML C++ Class Generator Example 3: CG Sample Program 3

Part IV Oracle XML Developer's Kit Reference

33 XSQL Pages Reference

33.1 XSQL Configuration File Parameters 2
33.2.1 <xsql:action> 6
33.2.2 <xsql:delete-request> 8
33.2.3 <xsql:dml> 9
33.2.4 <xsql:if-param> 10
33.2.5 <xsql:include-owa> 12
33.2.6 <xsql:include-param> 13
33.2.7 <xsql:include-posted-include-posted> 14
33.2.8 <xsql:include-request-params> 14
33.2.9 <xsql:include-xml> 16
33.2.10 <xsql:include-xsql> 17
33.2.11 <xsql:insert-param> 19
33.2.12 <xsql:insert-request> 20
33.2.13 <xsql:query> 21
33.2.14 <xsql:ref-cursor-function> 24
33.2.15 <xsql:set-cookie> 26
33.2.16 <xsql:set-page-param> 28

Copyright © 1999, 2025, Oracle and/or its affiliates.

33.2.17 <xsql:set-session-param> 30
33.2.18 <xsql:set-stylesheet-param> 32
33.2.19 <xsql:update-request> 33

34 Oracle XML Developer's Kit Standards

34.1 XML Standards Supported by XDK 1
34.1.1 Summary of XML Standards Supported by XDK 1
34.1.2 XML Standards for XDK for Java 2
34.1.2.1 DOM Standard for XDK for Java 2
34.1.2.2 XSLT Standard for XDK for Java 3
34.1.2.3 JAXB Standard for XDK for Java 3
34.1.2.4 Pipeline Definition Language Standard for XDK for Java 3
34.2 Character Sets Supported by XDK 4
34.2.1 Character Sets Supported by XDK for Java 4
34.2.2 Character Sets Supported by XDK for C 4

A XDK for Java XML Error Messages

A.1 XML Parser Error Messages A-1
A.2 DOM Error Messages A-10
A.3 XSLT Error Messages A-15
A.4 XPath Error Messages A-17
A.5 XML Schema Validation Error Messages A-22
A.6 Schema Representation Constraint Error Messages A-32
A.7 Schema Component Constraint Error Messages A-37
A.8 XSQL Server Pages Error Messages A-47
A.9 XML Pipeline Error Messages A-47
A.10 JAXB Error Messages A-49

B XDK for Java TXU Error Messages

B.1 DLF Error Messages B-1
B.2 TransX Informational Messages B-3
B.3 TransX Error Messages B-3
B.4 Assertion Error Messages B-4

C XDK for Java XSU Error Messages

C.1 Generic Error Messages C-1
C.2 Query Error Messages C-2

Copyright © 1999, 2025, Oracle and/or its affiliates.

C.3 DML Error Messages C-2

D Oracle XML Developer's Kit JavaBeans (Deprecated)

D.1 Introduction to XDK JavaBeans D-1
D.1.1 Prerequisites for Using XDK JavaBeans D-1
D.1.2 Standards and Specifications for XDK JavaBeans D-2
D.1.3 XDK JavaBeans Features D-2
D.1.3.1 DOMBuilder D-2
D.1.3.2 XSLTransformer D-2
D.1.3.3 DBAccess D-3
D.1.3.4 XMLDBAccess D-3
D.1.3.5 XMLDiff D-4
D.1.3.6 XMLCompress D-4
D.1.3.7 XSDValidator D-5
D.2 Using XDK JavaBeans: Overview D-5
D.2.1 Using XDK JavaBeans: Basic Process D-5
D.2.1.1 Using the DOMBuilder JavaBean: Basic Process D-5
D.2.1.2 Using the XSLTransformer JavaBean: Basic Process D-7
D.2.1.3 Using the XMLDBAccess JavaBean: Basic Process D-8
D.2.1.4 Using the XMLDiff JavaBean: Basic Process D-10
D.2.2 Running XDK JavaBean Demo Programs D-12
D.2.2.1 Running sample1 D-15
D.2.2.2 Running sample2 D-16
D.2.2.3 Running sample3 D-16
D.2.2.4 Running sample4 D-16
D.2.2.5 Running sample5 D-16
D.2.2.6 Running sample6 D-17
D.2.2.7 Running sample7 D-18
D.2.2.8 Running sample8 D-18
D.2.2.9 Running sample9 D-18
D.2.2.10 Running sample10 D-19
D.3 Processing XML with XDK JavaBeans D-19
D.3.1 Processing XML Asynchronously with the DOMBuilder and XSLTransformer Beans D-19
D.3.1.1 Parsing the Input XSLT Stylesheet D-21
D.3.1.2 Processing the XML Documents Asynchronously D-21
D.3.2 Comparing XML Documents with the XMLDiff JavaBean D-24
D.3.2.1 Comparing the XML Files and Generating a Stylesheet D-25

Copyright © 1999, 2025, Oracle and/or its affiliates.

Glossary

Index

Copyright © 1999, 2025, Oracle and/or its affiliates.

List of Examples

1-1	Oracle XML Developer's Kit Components	17
2-1	Improving Safety of Java Code that Uses an XML Parser	3
3-1	Oracle XML Developer's Kit for C Libraries, Header Files, Utilities, and Demos	2
3-2	Editing an Oracle XML Developer's Kit for C Make.bat File on Windows	8
5-1	NSExample.xml	10
5-2	xml.out	11
5-3	Using orastream Functions	14
5-4	XML Event Context	17
5-5	Sample Pull Parser Application Example	19
5-6	Sample Document to Parse	20
5-7	Events Generated by Parsing a Sample Document	20
5-8	Constructing a Schema-Based Document with the DOM API	23
5-9	Modifying a Database Document with the DOM API	25
7-1	Streaming Validator in Transparent Mode	6
7-2	Example of Streaming Validator in Opaque Mode	7
7-3	XmlSchemaLoad() Example	8
7-4	Example of Streaming Validator Using New Options	9
8-1	book1.xml	3
8-2	Sample Xdiff Instance Document	4
8-3	Xdiff Schema: xdiff.xsd	7
8-4	XMLDiff Application	9
8-5	Customized XMLDiff Output	11
8-6	Sample Application for XmlPatch	12
8-7	XmlHash Program	13
9-1	SOAP Request Message	2
9-2	SOAP Response Message	3
9-3	SOAP C Functions Defined in xmlsoap.h	5
9-4	Example 1 SOAP Message	7
9-5	Example 1 SOAP C Client	8
9-6	Example 2 SOAP Message	12
9-7	Example 2 SOAP C Client	12
9-8	Example 3 SOAP Message	14
9-9	Example 3 SOAP C Client	14
11-1	Oracle XML Developer's Kit for Java Libraries, Utilities, and Demos	2
11-2	Testing the Oracle XML Developer's Kit for Java Environment on UNIX	7
11-3	Testing the Oracle XML Developer's Kit for Java Environment on Windows	9

Copyright © 1999, 2025, Oracle and/or its affiliates.

11-4	XDKVersion.java	9
12-1	Sample XML Document	3
12-2	Sample XML Document Without Namespaces	8
12-3	Sample XML Document with Namespaces	8
12-4	Extracting Contents of a DOM Tree with selectNodes()	48
12-5	Incorrect Use of appendChild()	49
12-6	Merging Documents with appendChild	49
12-7	DTDSample.java	51
12-8	Converting XML in a String	54
12-9	Parsing a Document with Accented Characters	54
14-1	math.xml	7
14-2	math.xsl	7
14-3	math.htm	7
14-4	Using a Static Function in an XSLT Stylesheet	12
14-5	Using a Constructor in an XSLT Stylesheet	12
14-6	gettitle.xsl	13
14-7	msg_w_num.xml	14
14-8	msg_w_text.xml	14
14-9	msgmerge.xsl	15
14-10	msgmerge.xml	15
15-1	Simple Query Using XQJ	2
15-2	books.xml	3
15-3	books.xq	3
15-4	Executing a Query with a Custom Entity Resolver	3
15-5	trim.xq	4
15-6	Defining the Implementation of an External XQuery Function	5
15-7	Binding an External Function to a Java Static Method	6
15-8	math.xq	7
15-9	main.xq	7
15-10	Executing a Query that Imports a Library Module	7
15-11	size.xsd	9
15-12	size.xq	9
15-13	Executing an XQuery Query that Imports an XML Schema	9
15-14	Executing a Query with a Prefabricated File Resolver	11
15-15	Accessing the Values of Option Declarations	13
15-16	Using Option Declarations When Serializing a Query Result	13
15-17	books2.xq	15

Copyright © 1999, 2025, Oracle and/or its affiliates.

15-18	Facilitating Streaming Evaluation	16
15-19	Configuring the XQuery Processor to Use External Storage	17
15-20	configuration.xml	19
15-21	update.xq	19
15-22	Updated File configuration.xml	19
15-23	Executing the Updating Query update.xq	19
16-1	Using XQJ to Query an XML DB Table with XQuery	3
16-2	Using XQJ to Query the XML DB Repository with XQuery	4
17-1	family.dtd	2
17-2	family.xml	2
17-3	report.xml	3
17-4	report.xsd	4
17-5	Using oraxml to Validate Against a Schema	12
17-6	Using oraxml to Validate Against a DTD	12
18-1	sample3.xml	10
18-2	sample3.xsd	10
18-3	Address.java	11
18-4	sample10.xml	13
18-5	sample10.xsd	14
18-6	BusinessType.java	15
19-1	pipedoc.xml	9
20-1	Appending a Node	5
20-2	Inserting a Node	6
20-3	Deleting a Node	6
20-4	Getting a diff as a Document from a Java Application	8
20-5	Getting a diff Using DiffOpReceiver from a Java Application	8
20-6	Diff Output Schema: xdiff.xsd	10
21-1	Specifying skipRows and maxRows on the Command Line	16
21-2	upd_emp.xml	22
21-3	XSU-Generated Sample Document	28
21-4	customer.xml	31
21-5	createRelSchema.sql	31
22-1	Structure of Table translated_messages	10
22-2	Query of translated_messages	10
22-3	example.xml	10
22-4	example.xml with a Language Attribute	12
22-5	dateTime Row	13

Copyright © 1999, 2025, Oracle and/or its affiliates.

22-6	example_es.xml	14
22-7	example_es.xml with a Language Attribute	14
22-8	txdemo1.java	15
23-1	DLF Tree Structure	3
23-2	Minimal DLF Document	12
23-3	Sample DLF Document	12
23-4	DLF with Localization	14
24-1	Sample XSQL Page	1
24-2	Connection Definitions Section of XSQLConfig.xml	7
24-3	Sample XSQL Page in AvailableFlightsToday.xsql	13
24-4	Wrapping the <xsql:query> Element	13
24-5	Bind Variables in CustomerPortfolio.xsql	14
24-6	Bind Variables with Action Elements in CustomerPortfolio.xsql	14
24-7	Lexical Substitution Parameters for Rows and Columns in DevOpenBugs.xsql	15
24-8	Lexical Substitution Parameters for Connections and Stylesheets in DevOpenBugs.xsql	15
24-9	Setting a Default Value	17
24-10	Setting Multiple Default Values	17
24-11	Defaults for Bind Variables	17
24-12	Bind Variables with No Defaults	17
24-13	Industry Standard Formats in flight-list.xsl	21
24-14	Stylesheet Association in flight-list.xsl	21
24-15	Query Results in flight-display.xsl	23
24-16	XSQLRequestSample Class	24
24-17	Conditional Statements in XSQL Pages	26
24-18	Passing Values Among SQL Queries	26
24-19	Handling Multivalued Parameters	27
24-20	Using Multivalued Page Parameters in a SQL Statement	27
24-21	addmult PL/SQL Procedure	28
24-22	addmultwrapper PL/SQL Procedure	28
24-23	addmult.xsql	28
24-24	Getting the Name of the Current XSQL Page	29
25-1	empToExcel.xsl	3
25-2	emp_test.xsql	4
25-3	emp_test_dynamic.xsql	4
25-4	Multiple <?xml-stylesheet ?> Processing Instructions	5
25-5	Using an Array-Valued Parameter in an XSQL Page	6
25-6	testTableFunction	9

Copyright © 1999, 2025, Oracle and/or its affiliates.

25-7	XSQL Page with Array-Valued Parameters	9
25-8	Using an Array-Valued Parameter to Restrict Rows	10
25-9	Setting an Error Parameter	10
25-10	Achieving Conditional Behavior with an Error Parameter	11
25-11	XSLT Stylesheet	11
25-12	Aggregating a Dynamically-Constructed XML Document	12
25-13	Movie XML Document	13
25-14	Using XPath to Extract an Aggregate List	13
25-15	Including an XMLType Query Result	13
25-16	Using XSQL Bind Variables in an XPath Expression	14
25-17	XML Document Generated from HTML Form	16
25-18	Source Code for FOP Serializer	17
25-19	MyIncludeXSQLHandler.java	20
25-20	Testing for the Servlet Request	20
25-21	Custom Serializer	22
25-22	Assigning Short Names to Custom Serializers	23
25-23	Writing a Dynamic GIF Image	23
25-24	myErrorHandler class	26
25-25	SampleCustomLogger Class	27
25-26	SampleCustomLoggerFactory Class	27
25-27	Registering a Custom Logger Factory	27

Copyright © 1999, 2025, Oracle and/or its affiliates.

List of Figures

1-1	Sample XML Processor	3
1-2	XML Parsers for Java, C, and C++	4
1-3	Oracle JAXB Class Generator	6
1-4	XSU Processes SQL Queries and Returns the Result as XML	7
1-5	XSQL Pages Publishing Framework	9
1-6	XSLT Virtual Machine	10
1-7	Sample XML Processor Built with Java Oracle XML Developer's Kit Components	11
1-8	Generating XML Documents Using Oracle XML Developer's Kit C Components	12
1-9	Generating XML Documents Using Oracle XML Developer's Kit C++ Components	13
1-10	Oracle XML Developer's Kit Tools and Frameworks	14
3-1	The Property Pages	9
3-2	Setting the Include Path in Visual C++	10
3-3	Setting the Static Library Path in Visual C++	11
3-4	Setting the Names of the Libraries in Visual C++ Project	11
5-1	XML Parser for C Calling Sequence	5
7-1	XML Schema Processor for C Usage Diagram	4
11-1	Oracle XML Developer's Kit for Java Component Dependencies for JDK 5	3
12-1	XML Parser Process	3
12-2	Comparing DOM (Tree-Based) and SAX (Event-Based) APIs	7
12-3	XML Parser for Java	11
12-4	Basic Architecture of the DOM Parser	16
12-5	Using the SAXParser Class	34
12-6	SAX Parsing with JAXP	40
12-7	DOM Parsing with JAXP	41
13-1	Binary XML Encoding	8
13-2	Binary XML Decoder	8
14-1	Using the XSLT Processor for Java	4
17-1	XML Schema Processor for Java	8
18-1	JAXB Class Generator for Java	5
19-1	Pipeline Processing	2
19-2	Using the Pipeline Processor for Java	5
21-1	Generating XML with XSU	3
21-2	Storing XML in the Database Using XSU	4
21-3	Running XSU in the Database	6
21-4	Running XSU in the Middle Tier	7
21-5	Running XSU in a Web Server	7

Copyright © 1999, 2025, Oracle and/or its affiliates.

22-1	Basic Process of a TransX Application	3
24-1	XSQL Pages Framework Architecture	3
24-2	Web Access to XSQL Pages	4
24-3	XSQL Home Page	11
24-4	XML Result from XSQL Page (AvailableFlightsToday.xsql) Query	19
24-5	Exploring flight-list.dtd with XML Authority	20
24-6	XSQL Page Results in XML Format	21
24-7	Using an XSLT Stylesheet to Render HTML	22
D-1	DOMBuilder JavaBean Usage	D-6
D-2	XSLTransformer JavaBean Usage	D-8
D-3	XMLDBAccess JavaBean Usage	D-9
D-4	XMLDiff JavaBean Usage	D-11

Copyright © 1999, 2025, Oracle and/or its affiliates.

List of Tables

1-1	Overview of Oracle XML Developer's Kit Components	1
1-2	XDK for Java Components for Generating XML	11
3-1	Dependent Libraries of Oracle XML Developer's Kit for C Components on UNIX	3
3-2	UNIX Environment Settings for Oracle XML Developer's Kit for C Components	4
3-3	Oracle XML Developer's Kit for C/C++ Utilities on UNIX	4
3-4	Header Files in the Oracle XML Developer's Kit for C Compile-Time Environment	5
3-5	Dependent Libraries of Oracle XML Developer's Kit for C Components on Windows	6
3-6	Windows Environment Settings for Oracle XML Developer's Kit for C Components	7
3-7	Oracle XML Developer's Kit for C/C++ Utilities on Windows	7
3-8	Summary of Oracle XML Developer's Kit for C APIs	12
4-1	XSLT Processor for C: Command Line Options	5
4-2	XSLT for C Demo Files	6
5-1	Interfaces for XML, DOM, and SAX APIs	2
5-2	Data Types Used in the XML Parser for C	3
5-3	C Parser Demos	8
5-4	C XML Parser Command-Line Options	9
5-5	NULL-Terminated and Length-Encoded C API Functions	12
5-6	XMLType Functions	21
7-1	XML Schema Processor for C: Supplied Files in $ORACLE_HOME	2
7-2	XML Schema Processor for C: Supplied Libraries	2
7-3	XML Schema for C Samples Provided	4
8-1	XmlDiff Command-Line Options for the C Language	2
8-2	Xdiff Operation Attributes	5
8-3	XmlPatch for C Command-Line Options	12
10-1	Deprecated XDB Package Classes and Their Unified Java API Equivalents	2
10-2	Deprecated XMLType Methods and Their Unified Java API Equivalents	3
10-3	XMLDocument Output Based on KIND and CONNECTION	3
11-1	Java Libraries for Oracle XML Developer's Kit for Java Components	3
11-2	UNIX Environment Variables for Oracle XML Developer's Kit for Java Components	5
11-3	Oracle XML Developer's Kit for Java UNIX Utilities	6
11-4	Windows Environment Variables for Oracle XML Developer's Kit for Java Components	7
11-5	Oracle XML Developer's Kit for Java Windows Utilities	8
12-1	XML Parser for Java Validation Modes	9
12-2	XML Compression with DOM and SAX	10
12-3	Java Parser Demos	12
12-4	oraxml Command-Line Options	14

Copyright © 1999, 2025, Oracle and/or its affiliates.

12-5	DOMParser Configuration Methods	18
12-6	Some Interfaces Implemented by XMLDocument	19
12-7	Methods for Getting and Manipulating DOM Tree Nodes	19
12-8	ACCESS_MODE Attribute Values	24
12-9	Range Class Methods	29
12-10	Static Fields in the NodeFilter Interface	31
12-11	TreeWalker Interface Methods	31
12-12	SAX 2.0 Handler Interfaces	33
12-13	SAX 2.0 Helper Classes	33
12-14	SAXParser Methods for Registering Event Handlers	35
12-15	XMLTokenizer Methods	38
12-16	JAXP Packages	40
14-1	XSLT Processor Sample Files	4
14-2	Command-Line Options for oraxsl	6
14-3	XSLProcessor Methods	9
14-4	XMLDocumentFragment Methods	10
15-1	Descriptions of Various Types of Entity	12
15-2	XQJ Implementation-Defined Items	29
15-3	XQuery Implementation-Defined Items	30
15-4	XQuery Update Facility Implementation-Defined Items	32
15-5	Default Initial Values for the Static Context	32
16-1	OXQDDataSource Properties	5
16-2	Oracle XML DB Support for Optional XQJ Features	5
17-1	Feature Comparison Between XML Schema and DTD	5
17-2	oracle.xml.parser.schema Classes	7
17-3	XML Schema Sample Files	9
18-1	javax.xml.bind Classes and Interfaces	4
18-2	JAXB Class Generator Demos	6
18-3	orajaxb Command-Line Options	8
19-1	Methods in Class oracle.xml.pipeline.controller.Process	3
19-2	Classes in oracle.xml.pipeline.processes	4
19-3	PipelineProcessor Methods	6
19-4	Pipeline Processor Sample Files	7
19-5	orapipe Command-Line Options	9
19-6	PipelineErrorHandler Methods	12
21-1	XSU Sample Files	8
21-2	getXML Options	10

Copyright © 1999, 2025, Oracle and/or its affiliates.

21-3	putXML Options	11
22-1	TransX Utility Features	3
22-2	TransX Configuration Methods	4
22-3	TransX Utility Sample Files	5
22-4	TransX Utility Command-Line Options	7
22-5	TransX Utility Command-Line Parameters	8
22-6	<column> Attributes	11
22-7	date and dateTime Formats	12
23-1	Notation for Occurrence of Attributes and Elements	2
23-2	Entity References	5
23-3	DLF Elements	5
23-4	Top-Level Table Element	6
23-5	Translation Elements	6
23-6	Lookup Key Elements	6
23-7	Metadata Elements	7
23-8	Data Elements	8
23-9	Attributes	8
23-10	DLF Attributes	8
23-11	XML Namespace Attributes	12
24-1	XSQL Servlet Demos	8
25-1	Pseudo-Attributes for <?xml-stylesheet ?>	5
25-2	Helpful Methods in the XSQLActionHandlerImpl Class	18
26-1	Header Files in the XDK for C++ Compile-Time Environment	2
28-1	XML Parser for C++ Sample Files	5
29-1	XSLT for C++ Sample Files	3
30-1	XML Schema Processor for C++ Command-Line Options	2
30-2	XML Schema Processor for C++ Samples Provided	3
32-1	C++ Class Generator Options	1
32-2	XML C++ Class Generator Files	2
33-1	Built-In XSQL Elements and Action Handler Classes	1
33-2	XSQL Configuration File Settings	2
33-3	Attributes for <xsql:delete-request>	8
33-4	Attributes for <xsql:dml>	9
33-5	Attributes for <xsql:if-param>	11
33-6	Attributes for <xsql:include-owa>	12
33-7	Attributes for <xsql:include-xml>	16
33-8	Attributes for <xsql:include-xsql>	18

Copyright © 1999, 2025, Oracle and/or its affiliates.

33-9	Attributes for <xsql:insert-param>	19
33-10	Attributes for <xsql:insert-request>	21
33-11	Attributes for <xsql:query>	22
33-12	Attributes for <xsql:set-cookie>	26
33-13	Attributes for <xsql:set-page-param>	29
33-14	Attributes for <xsql:set-session-param>	31
33-15	Attributes for <xsql:set-stylesheet-param>	33
33-16	Attributes for <xsql:update-request>	34
34-1	Summary of XML Standards Supported by Oracle XML Developer's Kit	1
D-1	javax.xml.async DOM-Related Classes and Interfaces	D-5
D-2	javax.xml.async XSL-Related Classes and Interfaces	D-7
D-3	XMLDBAccess Methods	D-9
D-4	XMLDiff Methods	D-10
D-5	JavaBean Sample Java Source Files	D-12
D-6	JavaBean Sample Files	D-14

Copyright © 1999, 2025, Oracle and/or its affiliates.

Preface

This document describes the Oracle XML Developer's Kit (XDK). It provides detailed information about various language components, including Extensible Markup Language (XML), Java, C, and C++.

Audience

This document is for application developers who use the language components of the XDK to generate and store XML data in either a database or a document outside the database. Examples and sample applications are provided where possible. This document assumes familiarity with XML and either Java, C, or C++.

Related Documents

Oracle resources that are related to this document are listed.

For more information, see these resources:

- *Oracle XML DB Developer's Guide*
- *Oracle AI Database XML C API Reference*
- *Oracle AI Database XML C++ API Reference*
- *Oracle AI Database XML Java API Reference*
- *Oracle AI Database Transactional Event Queues and Advanced Queuing User's Guide*
- XDK on Oracle Technology Network

For additional information about XML, see:

- W3C XML specifications
- `XML.com`, a broad collection of XML resources and commentary
- *Annotated XML Specification*
- `XML.org`, hosted by OASIS as a resource to developers of purpose-built XML languages,

Examples

Many examples in this document use the Oracle Database sample database schemas or are otherwise provided with your software.

For information about how the sample schemas, see *Oracle Database Sample Schemas*.

Examples that are provided with the software can be found in these directories:

- `$ORACLE_HOME/xdk/demo/java/`

Copyright © 1999, 2025, Oracle and/or its affiliates.

- `$ORACLE_HOME/xdk/demo/c/`
- `$ORACLE_HOME/xdk/java/sample/`
- `$ORACLE_HOME/rdbms/demo`

Conventions

The text conventions that are used in this document are described.

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Changes in This Release

This is a summary of important changes in *Oracle XML Developer's Kit Programmer's Guide*.

Desupported Feature

The following is the desupported feature in *Oracle XML Developer's Kit Programmer's Guide* for Oracle AI Database Release 26ai.

Desupport of Oracle Database Extensions for .NET

Oracle Database Extensions for .NET is a feature of Oracle Database on Microsoft Windows that enables you to use stored procedures and functions written in a language managed by .NET, such as C#.

Oracle Database hosts the Microsoft Common Language Runtime (CLR) in an external process, outside of the Oracle Database process. Application developers can write stored procedures and functions using any .NET compliant language, such as C# and VB.NET, and use these .NET stored procedures in the database, in the same manner as other PL/SQL or Java stored procedures. .NET stored procedures can be called from PL/SQL packages, procedures, functions, and triggers; from SQL statements; or from anywhere a PL/SQL procedure or function can be called.

Migration options include:

- Moving the .NET code (assemblies) into a middle tier
- Using the External Procedures feature to have the external process load and execute the .NET assembly
- Rewriting the stored procedures using PL/SQL or Java

Deprecated Features

This section lists the deprecated features in *XML Developer's Kit Programmer's Guide* for Oracle AI Database Release 26ai.

Oracle recommends that you do not use deprecated features/values in new applications. Support for deprecated features is for backward compatibility only.

Deprecation of XML DB Repository

The Oracle XML DB Repository is deprecated with Oracle AI Database 26ai.

Oracle recommends that you replace any functionality used in XML DB Repository with alternative technologies. As a result of this deprecation, all XML DB Repository interfaces (for example Repository-specific Java classes `oracle.xdb.servlet`, `oracle.xdb.event`, and `oracle.xdb.spi`) are consequently deprecated as well.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Deprecation of DBMS_XMLGEN PL/SQL Package

The PL/SQL package `DBMS_XMLGEN` is deprecated in Oracle AI Database 26ai.

`DBMS_XMLGEN` is a non-standard Oracle-proprietary package that is provided to generate and convert XML documents from SQL queries or with PL/SQL. This package is deprecated, and can be desupported in a future release. Oracle recommends that you use SQL/XML operators to generate XML from relational columns instead. Using ANSI SQL/XML operators for any generation and modification of XML documents provides a standardized and future-proof way to work with XML documents.

Deprecation of Unstructured XML Indexes

Unstructured XML indexes are deprecated in Oracle AI Database 26ai.

Unstructured XML indexes are deprecated and superseded by XML search indexes. Oracle recommends that you use XML search indexes or structured XML indexes.

Copyright © 1999, 2025, Oracle and/or its affiliates.

1

Introduction to Oracle XML Developer's Kit

Oracle XML Developer's Kit (XDK) is introduced.

1.1 Overview of XDK

Oracle XML Developer's Kit (XDK) is a versatile set of components that enables you to build and deploy C, C++, and Java software programs that process Extensible Markup Language (XML). You can assemble these components into an XML application that serves your business needs.

> **Note**
>
> If you are using XDK with PL/SQL and migrating from Oracle Database Release 8.1 or 9.2, Oracle strongly recommends that you use database character set AL32UTF8. Otherwise, problems can arise when PL/SQL processes XML data that contains escaped entities.

Oracle XML Developer's Kit (XDK) supports Oracle XML DB, which is a set of technologies used for storing and processing XML in Oracle Database. You can use XDK with Oracle XML DB to build applications that run in Oracle Database. You can also use XDK independently of Oracle XML DB.

Dates and timestamps in generated XML are in the formats specified by XML Schema. See *Oracle XML DB Developer's Guide*.

XDK is fully supported by Oracle and comes with a commercial redistribution license. The standard installation of Oracle Database includes XDK.

Table 1-1 briefly describes the XDK components, tells which programming languages they support, and directs you to the sections of this document that explain how to use them.

Table 1-1 Overview of Oracle XML Developer's Kit Components

Component	Description	Languages	See
XML parser	Creates and parses XML with industry standard Simple API for XML (SAX) and Document Object Model (DOM) interfaces.	Java, C, C++	• XML Parsing for Java • Using the XML Parser for C • Using the XML Parser for C++
XML Compressor	Enables binary compression and decompression of XML documents. The XML compressor is built into the XML parser for Java.	Java	Compressing and Decompressing XML
Java API for XML Processing (JAXP)	Enables Java applications to use SAX, DOM, XML Schema processor, Extensible Stylesheet Language Transformations (XSLT) processors, or alternative processors.	Java	Parsing XML with JAXP

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 1-1 (Cont.) Overview of Oracle XML Developer's Kit Components

Component	Description	Languages	See
XSLT Processor	Transforms XML into other text-based formats such as Hypertext Markup Language (HTML).	Java, C, C++	• Using the XSLT Processor for Java • Using the XSLT and XVM Processors for C • Using the XSLT Processor for C++
XQuery Processor for Java	Enables Java applications to query, transform, and update XML directly in the Java Virtual Machine (JVM).	Java	Using the XQuery Processor for Java
XML Schema Processor	Validates schemas, allowing use of simple and complex XML data types.	Java, C, C++	• Using the XML Schema Processor for Java • Using the XML Schema Processor for C • Using the XML Schema Processor for C++
XML class generator	Generates Java or C++ classes from document type definitions (DTDs) or XML schemas so that you can send XML data from web forms or applications. The Java implementation supports Java Architecture for XML Binding (JAXB).	Java, C++	• Using the JAXB Class Generator • Using the XML Class Generator for C++
XML Pipeline Processor	Applies XML processes specified in a declarative XML Pipeline document.	Java	Using the XML Pipeline Processor for Java
XML JavaBeans	Provides bean encapsulations of XDK components for easy use of Integrated Development Environment (IDE), Java Server Pages (JSP), and applets.	Java	Oracle XML Developer's Kit JavaBeans (Deprecated)
XML Diffing Library for Java	Enables pure Java programs in the middle tier to exchange XMLDiff output with C programs or programs that use Oracle Database to perform XMLDiff operations.	Java	Determining XML Differences Using Java
XML SQL Utility (XSU)	Generates XML documents, DTDs, and Schemas from structured query language (SQL) queries. Maps any SQL query result to XML or the reverse. XSU Java classes are mirrored by PL/SQL packages.	Java, PL/SQL	Using the XML SQL Utility
TransX Utility	Loads translated seed data and messages into the database using XML.	Java	Using the TransX Utility
XSQL servlet	Combines XML, SQL, and XSLT in the server to deliver dynamic web content.	Java	Using the XSQL Pages Publishing Framework
Oracle SOAP Server	Provides a lightweight Simple Object Access Protocol (SOAP) messaging protocol for sending and receiving requests and responses across the Internet.	C	Using SOAP with the Oracle XML Developer's Kit for C
XSLT XVM Processor	Provides a high-performance XSLT transformation engine that supports compiled XSL stylesheets.	C, C++	XSLT XVM Processor

Copyright © 1999, 2025, Oracle and/or its affiliates.

See Also

- XDK Components for fuller descriptions of many components in Table 1-1
- Oracle XML Developer's Kit Standards to learn about XDK support for XML-related standards

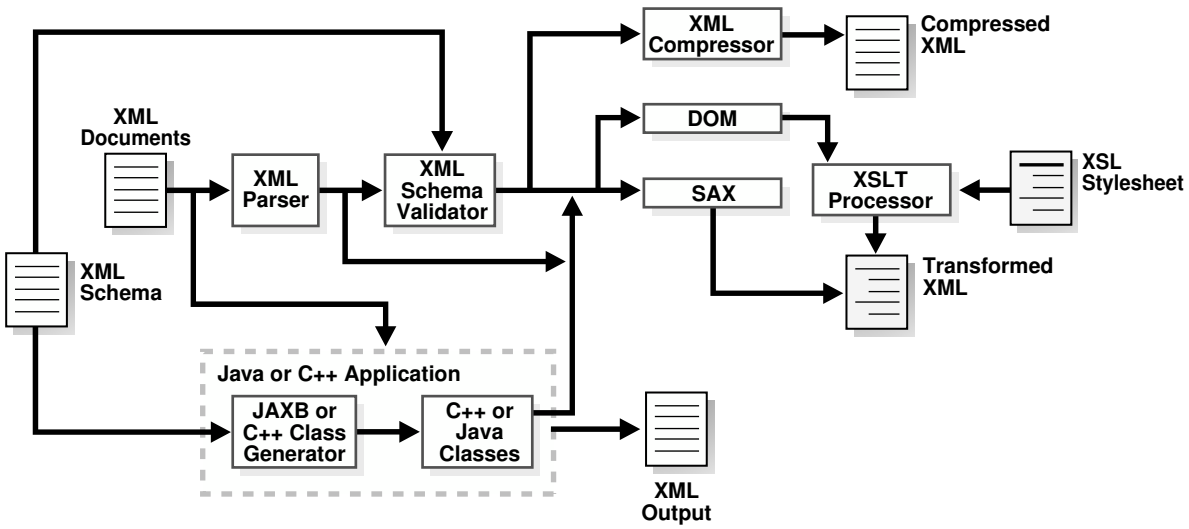
1.2 XDK Components

You can use XDK components in your programs to perform various types of XML processing.

Figure 1-1 shows a hypothetical XML processor that performs these tasks:

- Parse XML
- Validate XML against a DTD or XML schema
- Transform an XML document into another XML document by applying an XSLT stylesheet
- Generate Java and C++ classes from input XML schemas and DTDs

Figure 1-1 Sample XML Processor



1.2.1 XML Parsers

An XML parser reads an XML document and determines the structure and properties of the data. It breaks the data into parts and provides them to other XDK components.

An XML parser can programmatically access the parsed XML data with these APIs:

- SAX

  Use a SAX API to serially access the data element by element. You can register event handlers with a SAX parser and invoke callback methods when certain events are encountered.

- DOM

Copyright © 1999, 2025, Oracle and/or its affiliates.

Use a DOM API to represent the XML document as an in-memory tree and manipulate or navigate it.

XDK includes XML parsers for Java, C, and C++. Each parser includes support for both DOM and SAX APIs.

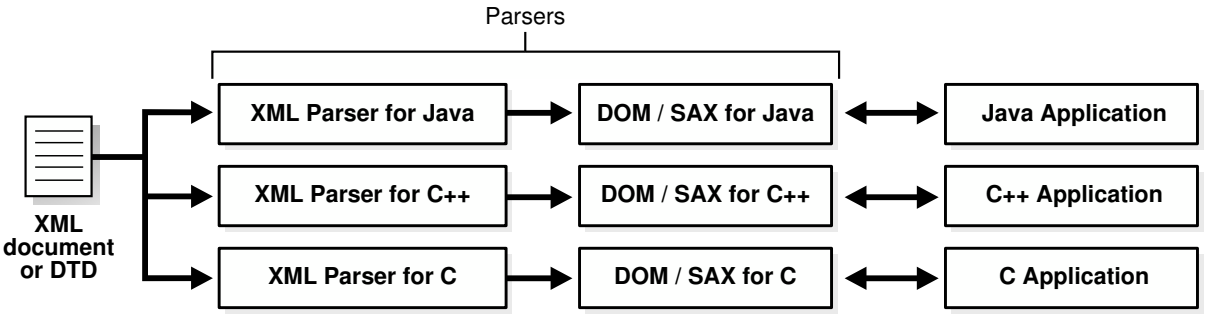
The XML parser for Java supports version 1.2 of Java API for XML Processing (JAXP), which is a standard API that enables use of DOM, SAX, XML Schema, and XSLT independently of a processor implementation. Thus, you can change the implementation of XML processors without impacting your programs.

The XML compressor is integrated into the XML parser for Java. It provides element-level XML compression and decompression with DOM and SAX interfaces. The XML compressor compresses XML documents without losing the structural and hierarchical information of the DOM tree. After parsing an XML document, you can serialize it with either DOM or SAX to a binary stream and then reconstruct it later.

You can use the XML compressor to reduce the size of XML message payloads, thereby increasing throughput. When used within applications as the internal XML document access, it significantly reduces memory usage while maintaining fast access.

Figure 1-2 shows the functionality of the XDK parsers for Java, C, and C++.

Figure 1-2 XML Parsers for Java, C, and C++



Related Topics

- XML Parsing for Java
  Extensible Markup Language (XML) parsing for Java is described.
- Using the XML Parser for C
  An explanation is given of how to use the Extensible Markup Language (XML) parser for C.
- Using the XML Parser for C++
  An explanation is given of how to use the Extensible Markup Language (XML) parser for C++.

1.2.2 XSLT Processors

XSLT is a stylesheet language that enables processors to transform one XML document into another. An XSLT document is a stylesheet that contains template rules that govern such a transformation. XDK enables XSLT transformation of XML data inside and outside the database on any operating system.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Each Oracle XML parser includes an integrated XSLT processor for transforming XML data using XSLT stylesheets. Using the XSLT processor, you can transform XML documents to XML, to Extensible Hypertext Markup Language (XHTML), or to almost any other text format.

Related Topics

- Using the XSLT Processor for Java
  An explanation is given of how to use the Extensible Stylesheet Language Transformation (XSLT) processor for Java.
- Using the XSLT and XVM Processors for C
  An explanation is given of how to use the Extensible Stylesheet Language Transformation (XSLT) and XSLT Virtual Machine (XVM) processors for C.
- Using the XSLT Processor for C++
  An explanation is given of how to use the Extensible Stylesheet Language Transformation (XSLT) processor for C++.

> **See Also**
>
> Specifications and other information are found on the W3C site at The Extensible Stylesheet Language Family (XSL)

1.2.3 XML Schema Processors

The XML Schema language, created by the W3C, describes the content and structure of XML documents in XML itself.

An XML schema contains rules that define validity for an XML application—this is its principal advantage over a DTD.

An XML schema specifies a set of built-in data types (such as string, float, and date). Users can derive their own data types from the built-in data types. For example, the schema can restrict dates to those after the year 2000 or specify a list of legal values.

XDK includes XML Schema processors for Java, C, and C++.

Related Topics

- Using the XML Schema Processor for Java
  Topics here cover how to use the Extensible Markup Language (XML) schema processor for Java.
- Using the XML Schema Processor for C
  An explanation is given of how to use the Extensible Markup Language (XML) schema processor for C.
- Using the XML Schema Processor for C++
  An explanation is given of how to use the Extensible Markup Language (XML) schema processor for C++.

1.2.4 XML Class Generators

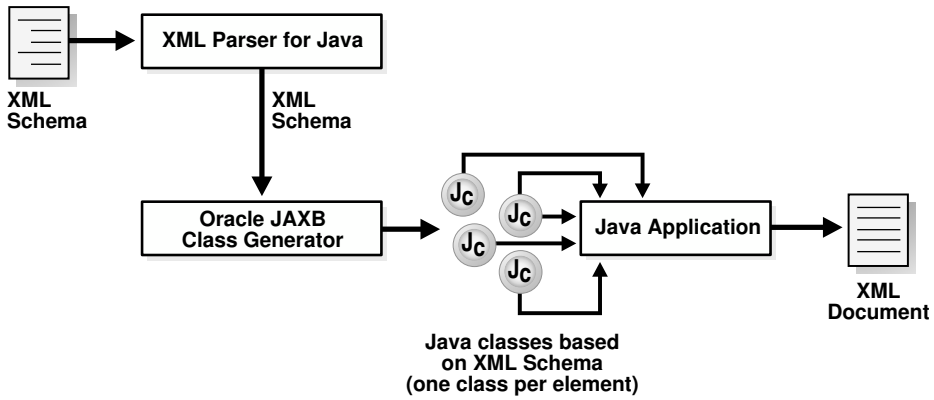
An XML class generator takes a parsed XML schema or DTD as input and generates Java or C++ source class files as output. XDK includes both the Java Architecture for XML Binding (JAXB) class generator and the C++ class generator.

Copyright © 1999, 2025, Oracle and/or its affiliates.

JAXB is a Java API and set of tools that maps XML data to Java objects, and the reverse. Because JAXB presents an XML document to a Java program in a Java format, you can write Java programs that process XML data without using a SAX parser or writing callback methods. Each Java object derives from an instance of the schema component in the input XML document. JAXB does not directly support DTDs, but you can convert a DTD to an XML schema that JAXB can use. The XML class generator for C++ directly supports both DTDs and XML Schemas.

As an example of how to use JAXB, you can write a Java program that uses generated Java classes to build XML documents gradually. Suppose that you write an XML schema for use by a human resources department and a Java program that responds to users who change their personal data. The program can use JAXB to construct an XML confirmation document in a piecemeal fashion, which an XSLT processor can transform into XHTML and deliver to a browser.

Figure 1-3 Oracle JAXB Class Generator



Related Topics

- Using the JAXB Class Generator
  An explanation is given of how to use the Java Architecture for XML Binding (JAXB) class generator.
- Using the XML Class Generator for C++
  Topics here explain how to use the Extensible Markup Language (XML) class generator for C++.

1.2.5 XML Pipeline Processor

The XML Pipeline Definition Language is an XML vocabulary for describing the processing relationships between XML resources. Oracle XML Pipeline processor conforms to the XML Pipeline Definition Language 1.0 standard.

The XML Pipeline processor takes as input an XML pipeline document (which defines the relationship between processes) and executes the pipeline processes according to the derived dependencies. For example, the input document can specify that the program must first validate an input XML document and then, if it is valid, transform it.

The XML pipeline processor helps Java developers by replacing custom Java code with a simple declarative XML syntax for building XML processing applications.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Related Topics

- Using the XML Pipeline Processor for Java
  An explanation is given of how to use the Extensible Markup Language (XML) pipeline processor for Java.

1.2.6 Oracle XML SQL Utility

Oracle XML SQL Utility (XSU) is a set of Java class libraries that you can use to render the results of SQL queries into canonical XML or to load data from an XML document into an existing database schema or view.

You can use XSU for these tasks:

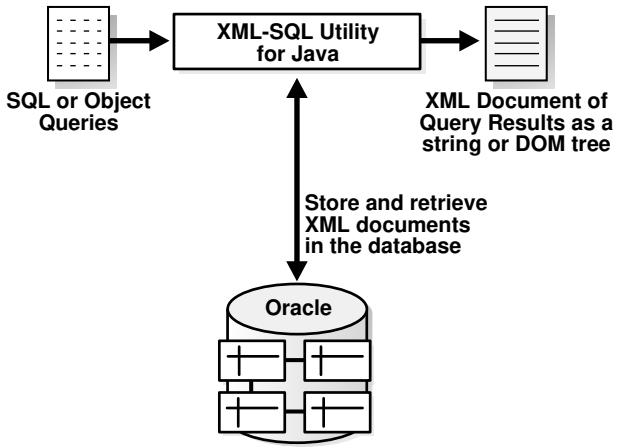
- Automatically and dynamically render the results of arbitrary SQL queries into canonical XML.

  XSU supports queries over richly structured, user-defined object types and object views, including `XMLType`. XSU transforms relational data into XML like this:

  - Columns become top-level elements.
  - Scalar values become elements with text-only content.
  - Object types become elements with attributes appearing as subelements.
  - Collections are mapped to lists of elements.
- Load data from an XML document into an existing database schema or view.

Figure 1-4 shows how XSU processes SQL queries and returns the results as an XML document.

Figure 1-4 XSU Processes SQL Queries and Returns the Result as XML



1.2.6.1 XML Document Representations

XSU representations in which you can generate an XML document are described, along with their typical use cases.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML Document Representation	When to Use This Representation
String	When returning the XML document to a requester
In-memory DOM tree	When operating on the XML programmatically (for example, when transforming it with the XSLT processor by using DOM methods to search or modify the XML)
Series of SAX events	When retrieving XML, especially large documents or result sets

1.2.6.2 Using XSU with an XML Class Generator

You can use XSU to generate an XML schema that is based on the relational schema of an underlying table or view that you are querying. You can use the generated XML schema as input to the JAXB class generator or the C++ class generator.

You can then write code that uses the generated classes to create the infrastructure behind a web-based form. Based on this infrastructure, the form can capture user data and create an XML document compatible with the database schema. A program can write the XML directly to the corresponding table or object view without further processing.

Related Topics

- Using the XML SQL Utility
  An explanation is given of how to use the Extensible Markup Language (XML) SQL Utility (XSU).

1.2.7 TransX Utility Overview

The Oracle TransX utility enables you to populate a database with multilingual XML data. The utility uses a data format that is intuitive for both developers and translators and uses a validation capability that is less error-prone than previous techniques.

The TransX utility is an application of XSU that loads translated seed data and messages into a database schema. For populating a database with data in multiple languages, the TransX utility provides functionality that you would otherwise have to develop with XSU.

Related Topics

- Using the TransX Utility
  An explanation is given of how to use the TransX utility to transfer XML data to a database.

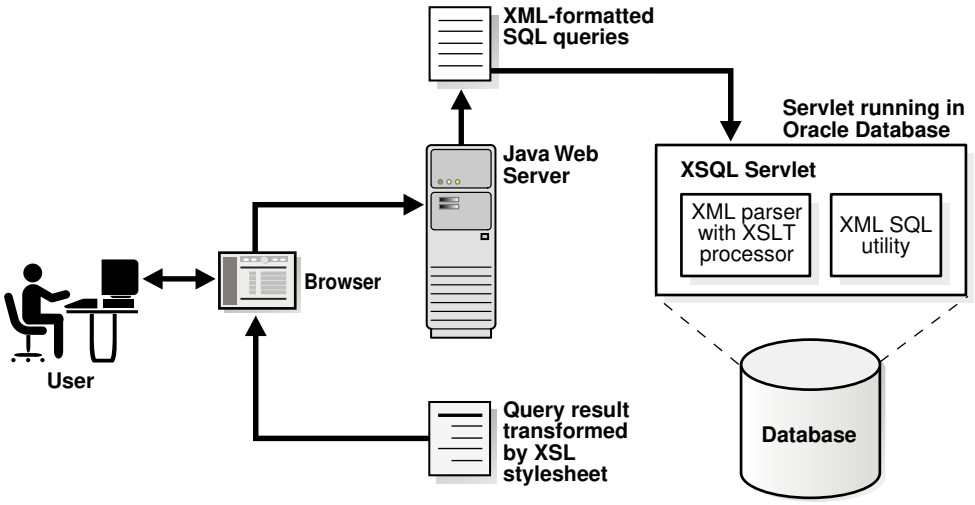
1.2.8 XSQL Pages Publishing Framework

The XSQL pages publishing framework (XSQL servlet) is a server component that takes an XSQL file (an XML file with a specific structure and grammar) and produces dynamic XML documents from one or more SQL queries of data objects.

Figure 1-5 shows how you can invoke the XSQL servlet.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Figure 1-5 XSQL Pages Publishing Framework



The XSQL servlet uses the Oracle XML parser to process the XSQL file, passing XSLT processing statements to its internal processor while passing parameters and SQL queries between the tags to XSU. Results from those queries are received as XML-formatted text or a Java Database Connectivity (JDBC) `ResultSet` object. If necessary, you can further transform the query results by using the built-in XSLT processor.

One example of an XSQL servlet is a page that contains a query of flight schedules for an airline with a bind variable for the airport name. The user can pass an airport name as a parameter in a web form. The servlet binds the parameter value in its database query and transforms the output XML into HTML for display in a browser.

Related Topics

- Using the XSQL Pages Publishing Framework
  An explanation is given of how to use the basic features of the XSQL pages publishing framework.

1.2.9 SOAP Services

Simple Object Access Protocol (SOAP) is a platform-independent messaging protocol that lets programs access services, objects, and servers. Oracle SOAP Services is published and executed through the web. It provides the standard XML message format for all programs.

SOAP Services lets you use XDK to develop messaging, remote procedure calls (RPC), and web service programs using XML standards.

Related Topics

- Using SOAP with the Oracle XML Developer's Kit for C
  An explanation is given of how to use Simple Object Access Protocol (SOAP) with the Oracle XML Developer's Kit (XDK) for C.

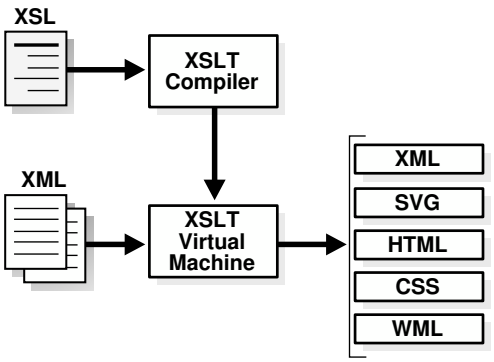
Copyright © 1999, 2025, Oracle and/or its affiliates.

1.2.10 XSLT Virtual Machine

The XSLT Virtual Machine (XVM) for C/C++ is the software implementation of a CPU designed to run compiled XSLT code. To run this code, you must compile XSLT stylesheets into byte code that the XVM engine understands.

Figure 1-6 shows how the XVM processes XML and XSL.

Figure 1-6 XSLT Virtual Machine



XDK includes an XSLT compiler that is compliant with the XSLT 1.0 standard. The compilation can occur at runtime or be stored for runtime retrieval. Applications perform transformations faster, and with higher throughput, because the stylesheet does not need parsing and the templates are applied using an index lookup instead of an XML operation.

1.3 Generating XML Documents Using XDK

XDK lets you map the structure of an XML document to a relational schema. You can use XDK to create XML documents from database tables and insert XML-tagged data into tables. Each XDK programming language supports the development of programs that generate XML documents from relational data.

1.3.1 XML Document Generation with Java

The XDK components for generating XML documents with Java are XSL, XSU, JDBC, JAXB, JavaBeans, and XSLT.

Figure 1-7 shows how to use XDK for Java components to generate XML documents from relational data. For generating an XML document from a SQL query, you have a choice of three components, which are labeled A, B, and C. The components that your program can use to further process the XML document are labeled D, E, and F.

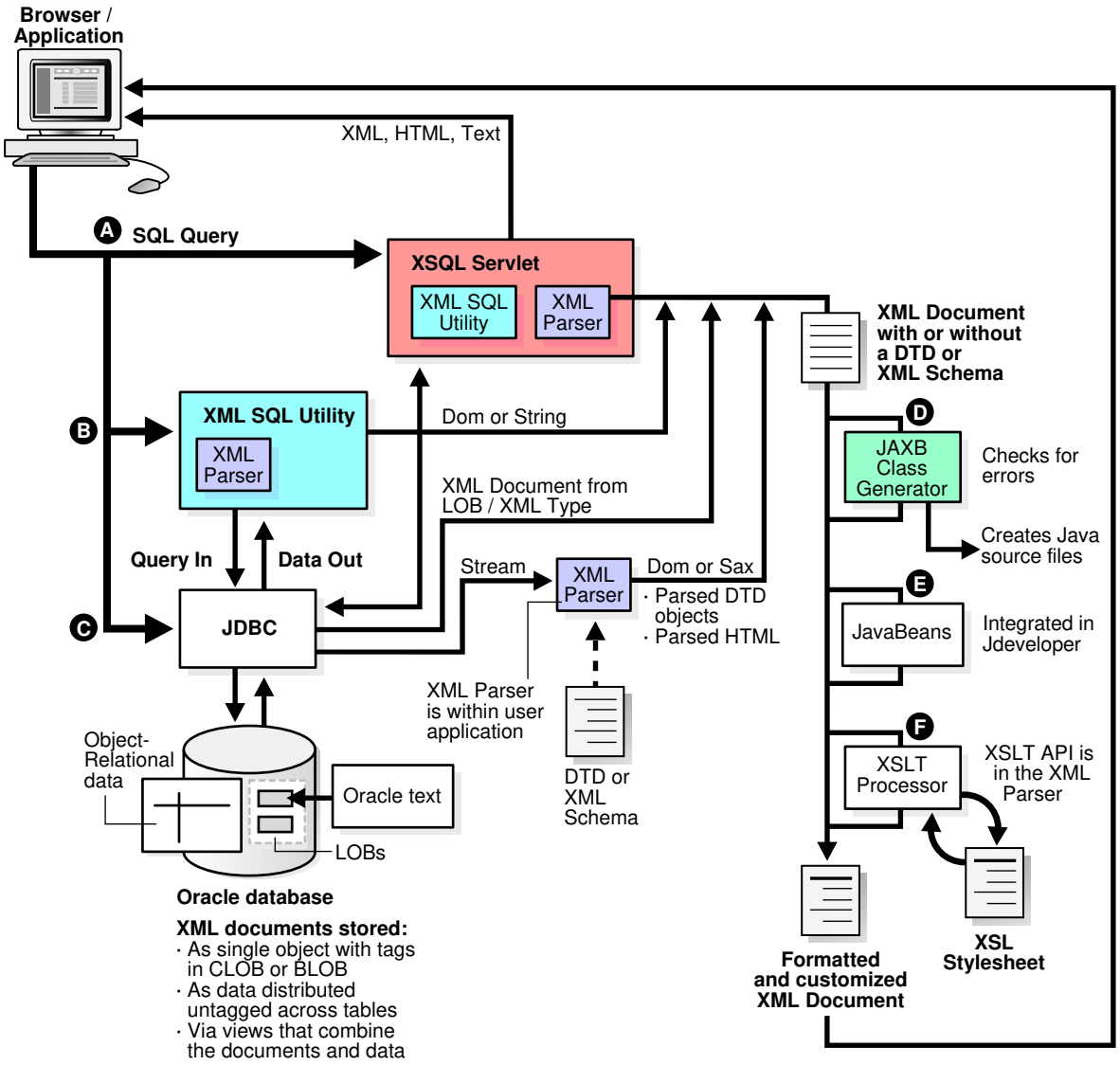
Table 1-2 describes the XDK for Java components.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 1-2 XDK for Java Components for Generating XML

Component	Label in Figure 1-7	Description
XSQL Servlet	A	Includes XSU and the XML parser
XSU	B	Includes XML parser
JDBC	C	Sends output data to the XML parser
JAXB	D	Generates Java class files that correspond to an input XML Schema
JavaBeans	E	Can compare an XML document with another XML document
XSLT	F	Transforms the XML document into XHTML with an XSLT stylesheet

Figure 1-7 Sample XML Processor Built with Java Oracle XML Developer's Kit Components



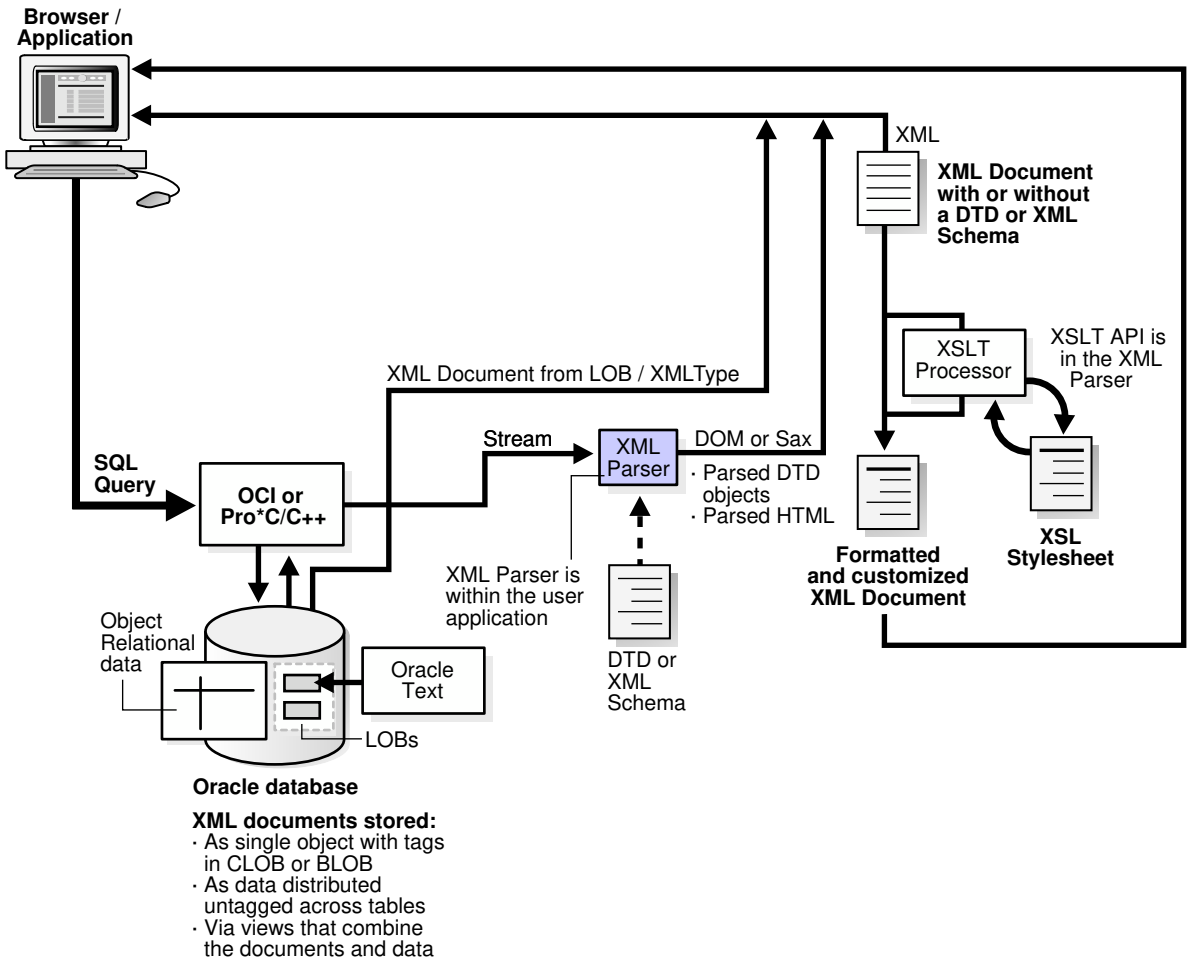
Copyright © 1999, 2025, Oracle and/or its affiliates.

1.3.2 XML Document Generation with C

An overview is presented of generating XML documents using XDK for C components.

Figure 1-8 shows how to use XDK for C components to generate XML documents from relational data. For component descriptions, see Table 1-1.

Figure 1-8 Generating XML Documents Using Oracle XML Developer's Kit C Components



To develop a C program that processes an XML document:

1. Send SQL queries to the database by using either the Oracle Call Interface (OCI) or Pro*C/C++ Precompiler. Your program must leverage the Oracle XML DB XML view functionality.
2. Process the resulting XML data with the XML parser or from the CLOB as an XML document.
3. Either transform the XML document with the XSLT processor, send it to an XML-enabled browser, or send it to a software program for further processing.

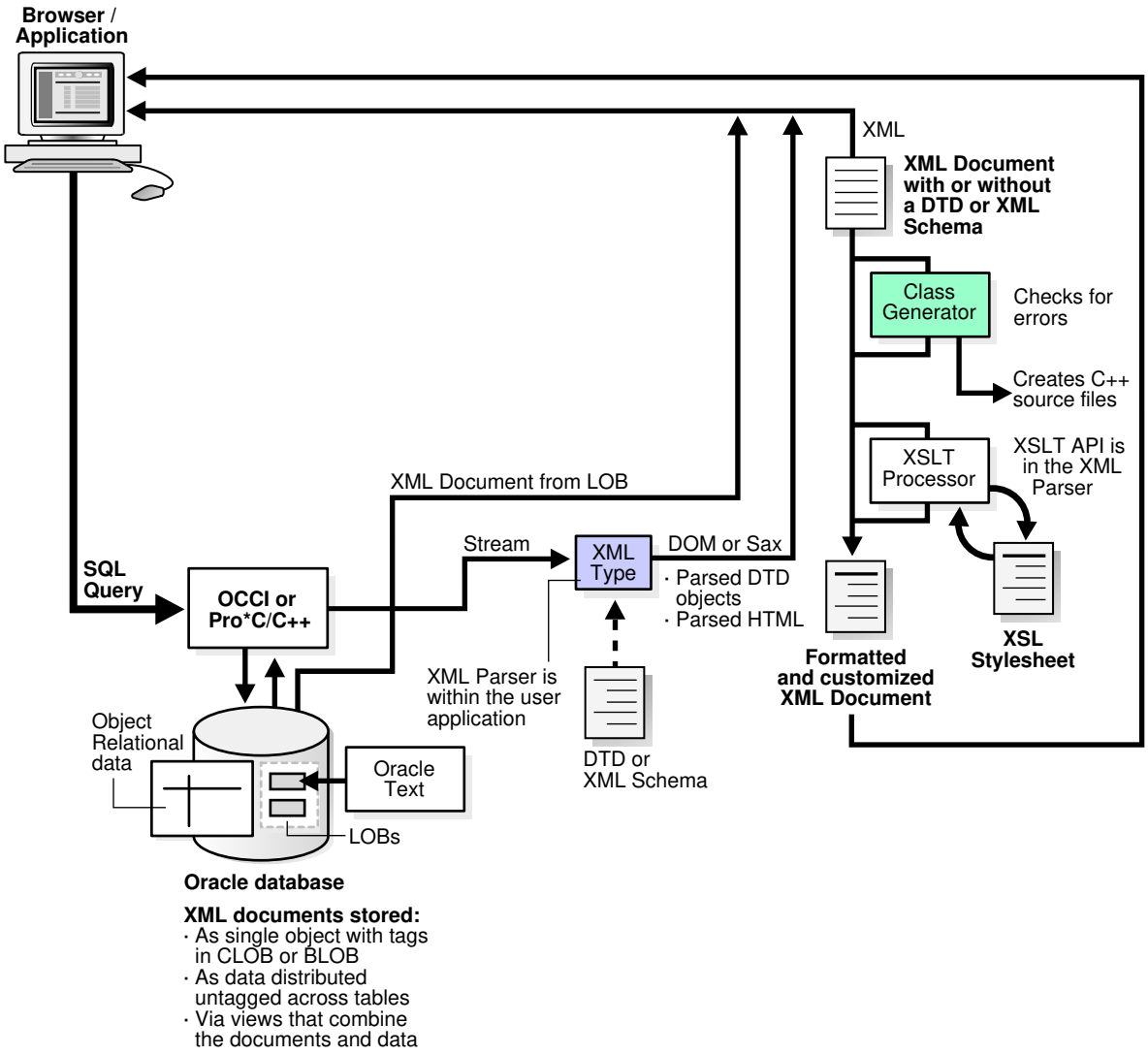
Copyright © 1999, 2025, Oracle and/or its affiliates.

1.3.3 XML Document Generation with C++

An overview is presented of generating XML documents using XDK for C++ components.

Figure 1-9 shows how to use XDK for C++ components to generate XML documents from relational data. For component descriptions, see Table 1-1.

Figure 1-9 Generating XML Documents Using Oracle XML Developer's Kit C++ Components



To develop a C++ program that processes an XML document:

1. Send SQL queries to the database by using either the Oracle C++ Call Interface (OCCI) or the Pro*C/C++ Precompiler.
2. Process the resulting XML data with the XML parser or from the CLOB as an XML document.
3. Either transform the XML document with the XSLT processor, send it to an XML-enabled browser, or send it to a software program for further processing.

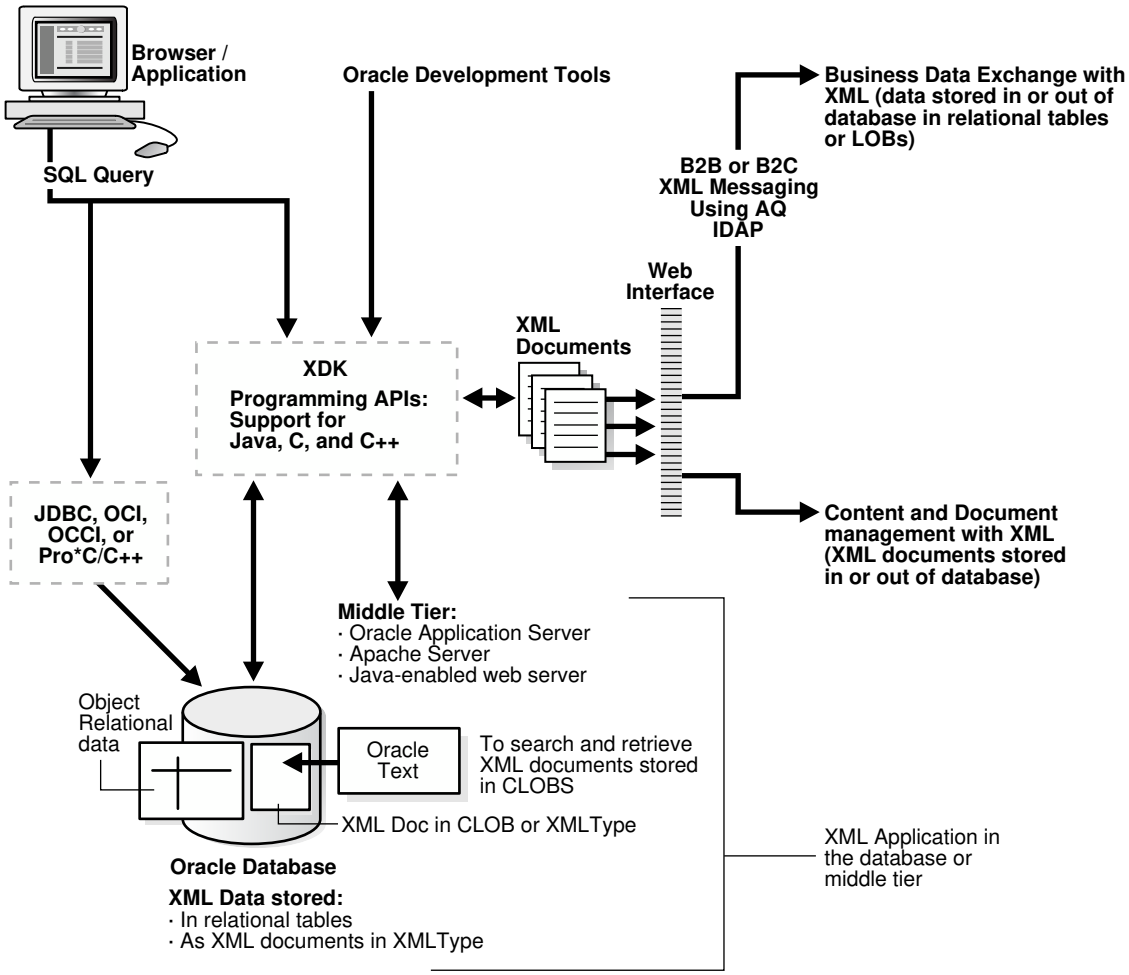
Copyright © 1999, 2025, Oracle and/or its affiliates.

1.4 Development Tools and Frameworks for XDK

Some tools and frameworks that you can use to develop software programs that use XDK components are presented.

Figure 1-10 illustrates this.

Figure 1-10 Oracle XML Developer's Kit Tools and Frameworks



For example, you can use Oracle JDeveloper to write a Java client that can query the database, generate XML, and perform additional processing. An employee can then use this program to send a query to Oracle Database. The program can transfer XML documents to XML-based business solutions for data exchange with other users, content and data management, and so forth.

1.4.1 Oracle JDeveloper

Oracle JDeveloper is a Java Platform, Enterprise Edition (Java EE) development environment with end-to-end support for developing, debugging, and deploying e-business applications. It provides a comprehensive set of integrated tools that support the complete development life

Copyright © 1999, 2025, Oracle and/or its affiliates.

cycle, from source code control, modeling, and coding through debugging, testing, profiling, and deployment.

Oracle JDeveloper simplifies development by providing deployment tools to create Java EE components such as:

- Applets
- JavaBeans
- Java Server Pages (JSP)
- Servlets
- Enterprise JavaBeans (EJB)

Oracle JDeveloper also provides a public API to extend and customize the development environment and integrate it with external products.

XDK is integrated into Oracle JDeveloper, offering many ways to manage XML. For example, you can use the XSQL Servlet to perform these tasks:

- Query and manipulate database information
- Generate XML documents
- Transform XML with XSLT stylesheets
- Deliver XML on the web

Oracle JDeveloper has an integrated XML schema-driven code editor for working on XML Schema-based documents such as XML schemas and XSLT stylesheets. By specifying the schema for a certain language, the editor can help you create a document in that markup language. The Code Insight feature can list valid alternatives for XML elements or attributes in the document.

Oracle JDeveloper simplifies the task of working simultaneously with Java application code and XML data and documents. It features drag-and-drop XML development modules such as:

- Color-coded syntax highlighting for XML
- Built-in syntax checking for XML and XSL
- Editing support for XML schema documents
- XSQL Pages and Servlet support
- Oracle XML parser for Java
- XSLT processor
- XDK for JavaBeans components
- XSQL Page Wizard
- XSQL Action Handlers
- Schema-driven XML editor

See Also

Oracle JDeveloper on OTN for links to Oracle JDeveloper documentation and tutorials

Copyright © 1999, 2025, Oracle and/or its affiliates.

1.4.2 Oracle Data Provider for .NET

Oracle Data Provider for .NET (ODP.NET) is an implementation of a .NET data provider for Oracle Database. It uses Oracle native APIs to provide fast, reliable access to Oracle data and features from any .NET application. It uses and inherits classes and interfaces from the Microsoft .NET Framework Class Library.

You can use ODP.NET and XDK to extract data from relational and object-relational tables and views as XML documents. You can also use XML documents for insert, update, and delete operations on the database server. ODP.NET supports XML natively in the database through Oracle XML DB.

ODP.NET supports XML with features that:

- Store XML data natively in the database server as the Oracle native type `XMLType`.
- Access relational and object-relational data as XML data from database instances into a Microsoft .NET environment and process the XML with the Microsoft .NET framework.
- Save changes to the database server with XML data.

For the ODP.NET application developer, features include:

- Enhancements to the `OracleCommand`, `OracleConnection`, and `OracleDataReader` classes
- XML-specific classes:
  - `OracleXmlType`
  - `OracleXmlStream`
  - `OracleXmlQueryProperties`
  - `OracleXmlSaveProperties`

> **See Also**
>
> *Oracle Data Provider for .NET Developer's Guide*

1.5 About Installing XDK

The standard installation of Oracle Database includes XDK (all of its components).

> **Caution**
>
> Using the components of Oracle XML Developer's Kit (XDK) to build software programs enables some powerful but potentially dangerous features, such as external entity expansion and recursive expansion. Refer to Security Considerations for Oracle XML Developer's Kit for information about how to use XDK securely.

This section assumes that:

- You installed Oracle Database from either a CD-ROM or an archive that you downloaded from the Oracle Technology Network (OTN).

Copyright © 1999, 2025, Oracle and/or its affiliates.

The Oracle Database CD-ROM installs XDK by default.

- You installed the XDK demo programs from the Oracle Database Examples media.

Example 1-1 shows how your Oracle Database home directory looks after you have installed Oracle Database and the XDK demo programs.

The directory that contains XDK is called **XDK home**. Set the value of environment variable `$XDK_HOME` (UNIX) or `%XDK_HOME%` (Windows) to the `xdk` directory in your Oracle home directory. For example, you can use `csh` on UNIX to set the XDK home directory with this command:

```
setenv XDK_HOME $ORACLE_HOME/xdk
```

Example 1-1 Oracle XML Developer's Kit Components

```
- $ORACLE_HOME: Oracle home directory
   | - bin: includes XDK executables
   | - lib: includes XDK libraries
   | - jlib: includes Globalization Support libraries for the XDK
   | - nls: includes binary files used as part of globalization support
   | - xdk: XDK scripts, message files, documentation, and demos
        readme.html
        | - admin: SQL scripts and XSL Servlet Configuration
                   file (XSQLConfig.xml)
        | - demo: sample programs (installed from Oracle Database Examples
media)
             | - c
             | - cpp
             | - java
             | - jsp
        | - doc: release notes and readme
             content.html
             index.html
             license.html
             title.html
              | - cpp
              | - images
              | - java
        | - include: header files
        | - mesg: error message files
```

Related Topics

- Installing XDK for Java Components
  XDK for Java components are included with Oracle Database. This chapter assumes that you installed XDK with Oracle Database and installed the demo programs from the Oracle Database Examples media.
- Installing XDK for C Components
  XDK for C components are the building blocks for reading, manipulating, transforming, and validating Extensible Markup Language (XML). The XDK for C components are included with Oracle Database.
- Installing XDK for C++ Components
  The XDK for C++ components are included with Oracle Database.

Copyright © 1999, 2025, Oracle and/or its affiliates.

2

Security Considerations for Oracle XML Developer's Kit

The security measures to be taken when using software programs that are built using XDK are explained in this chapter.

2.1 Implementing Security for Java

The process to implement security for Java programs.

2.1.1 Securing XSLT Processing with Oracle XML Developer's Kit

Before using Oracle XML Developer's Kit for XSLT processing, you need to secure it by setting some configuration options.

You must enable secure processing for both the `XSLProcessor` API and the JAXP transformer API, as follows.

- Use this Java code to enable the secure processing mode for `XSLProcessor`:

  ```
  processor.setAttribute(XMLConstants.FEATURE_SECURE_PROCESSING,
  Boolean.TRUE);
  ```

- Use this Java code to enable the secure processing mode for `SAXTransformerFactory` of the JAXP transformer API:

  ```
  factory.setFeature(XMLConstants.FEATURE_SECURE_PROCESSING, true);
  ```

Together, those settings improve XML-parsing security in the following ways:

- The settings block access to external resources, by default.

  Developers can then use `org.xml.sax.EntityResolver` or `javax.xml.transform.URIResolver` to resolve `include` and `import` elements in an XSL stylesheet. They can use Java interface `URIResolver` to provide XPath or XSLT functions access to trusted external resources.

- The settings block the use of Java reflection to execute Java code.

  Developers can convert trusted Java classes to interface `javax.xml.xpath.XPathFunction` and use interface `javax.xml.xpath.XPathFunctionResolver` to resolve them.

- The settings limit recursive entity expansion.

  This helps avoid a common denial-of-service attack from excessive resource consumption.

Copyright © 1999, 2025, Oracle and/or its affiliates.

> **Note**
>
> Although this can limit recursive entity *expansion*, XDK cannot limit resource consumption during XML transformation. An XSL stylesheet can include an infinite loop or otherwise consume resources extravagantly. For this reason, XSL stylesheets must not be accepted from untrusted sources or external entities. *Test all stylesheets used*, to ensure that they do not consume excessive resources.

In addition, Oracle recommends that all code that performs XSLT processing of data or files obtained from users or from external, untrusted entities check that secure processing is enabled for both the XSLProcessor API and the JAXP transformer API. Not doing this opens a security vulnerability. After the XSL processor has been secured, accessing external resources and running Java extension functions is not allowed.

Arbitrary Security Exemptions

If you want to arbitrarily override security restrictions, you can use the following options:

- Perform any of the following modifications to allow access to some external sources:
  - Set `URIResolver` to either `SAXTransformerFactory` or `XSLProcessor`, using the method `setURIResolver()`.
  - Set `EntityResolver` to `XSLProcessor` using the method `setEntityResolver()`.
- Use the Java interface `oracle.xml.xslt.XSLSecurityManager` to register a list of Java classes and methods with which you can use Java reflection extension functions.
- Register `XSLSecurityManager` using the `XSLProcessor` method `setAttribute()` to run an extension function that you know to be safe:

  ```
  processor.setAttribute(XSLProcessor.SECURITY_MANAGER, securityManager);
  ```

- Use a simple API on `XSLSecurityManager` to implement a the list of Java classes and methods:

  ```
  boolean checkExtensionFunction(String className, String fnName);
  ```

2.1.2 Using the Oracle XML Parser Safely

Like many XML parsers, by default the XDK parser tries to resolve external references. An attacker can exploit this behavior to perform XML External Entity (XXE) and XML Entity Expansion (XEE) attacks. To avoid this vulnerability, disable the use of arbitrary references to external resources, and disable unconstrained entity expansion.

- To implement a global security setting, which blocks all parsers that are created within the system from using external entities, set the Java system property `oracle.xdkjava.security.resolveEntityDefault` to `false`.
- To block entity expansion for a particular parser, or to limit the number of levels of entity expansion, set the attribute `XMLParser.RESOLVE_ENTITY_DEFAULT`, or `XMLParser.ENTITY_EXPANSION_DEPTH` to the parser.
  For example:

  ```
  DOMParser domParser = new DOMParser(); // Extend
  oracle.xml.parser.v2.XMLParser
  domParser.setAttribute(DOMParser.EXPAND_ENTITYREF, false); // Do
  ```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
not expand entity references
domParser.setAttribute(DOMParser.DTD_OBJECT, dtdObj);  // dtdObj is an
instance of oracle.xml.parser.v2.DTD
domParser.setAttribute(DOMParser.ENTITY_EXPANSION_DEPTH, 2); // Allow only
one level of recursive entity expansion
```

- Alternately, you can run `setSecureProcessing()` which sets the `EXPAND_ENTITYREF`, `DTD_OBJECT` and `ENTITY_EXPANSION_DEPTH` attributes by default.

  ```
  domParser.setSecureProcessing()// Set all the above attributes
  ```

- If you use JAXP to enable XML parsing, then set `FEATURE_SECURE_PROCESSING` to `true` for `DocumentBuilderFactory` or `SAXParserFactory`. This blocks the expansion of external entities, limits entity expansion to a depth of 11, and limits the entity expansion count to 64000.

  ```
  my_factory.setFeature(XMLConstants.FEATURE_SECURE_PROCESSING, true);
  ```

> **See Also**
>
> XML External Entity (XXE) Prevention Cheat Sheet

Example 2-1 Improving Safety of Java Code that Uses an XML Parser

This Java snippet blocks resolution of external entities, limits the number of entity-expansion levels, and limits the number of entity expansions.

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
dbf.setNamespaceAware(true);
dbf.setXIncludeAware(false);
dbf.setExpandEntityReferences(false);  // Disable expanding of entities
try {
  dbf.setFeature(XMLConstants.FEATURE_SECURE_PROCESSING, true);
} catch (Throwable e) {
    //handle old parser version
    ...
}
DocumentBuilder dp = dbf.newDocumentBuilder();
// Make resolveEntity throw an exception
//   (If you need to support external references,
//    then resolve publicId only against trusted values)
dp.setEntityResolver(new EntityResolver() {
  public InputSource resolveEntity(String publicId, String systemId)
                       throws DOMException, IOException {
    throw new DOMException((short)0, "Security Violation");
  }
});
...
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

2.2 Implementing Security for C

The process to implement security for C programs.

Block Insecure Accesses

Usually, there are two sets of XDK C APIs that take the risks of the XXE attack: one set uses the function `LpxInitEncoded()`and the other set uses the function `XmlCreate()`or `XmlCreateNew()`

For `LpxInitEncoded()`, feed the flags `LPX_FLAG_URL_DONT_OPEN` and `LPX_FLAG_FILE_DONT_OPEN` into the parser API to block external resource accesses.

For example:

```
LpxBufferParse ( ctx,
                 (oratext *) buf,
                 (size_t) lstlen ( buf ),
                 (oratext *) "UTF-8", (lx_langid) 0, (lxglo *) 0,
                 LPX_FLAG_URL_DONT_OPEN | LPX_FLAG_FILE_DONT_OPEN);
```

For `XmlCreate()`or `XmlCreateNew()`, set the parameter `no_ri_open` to `TRUE` when creating the context `xctx`. This setting will block insecure accesses for parser APIs which use `xctx`.

For example:

```
xctx = XmlCreateNew(&xerr, (oratext *) "test",
                    (oratext **)NULL, 0, (oratext *)NULL,
                    "data_encoding", “AL32UTF8”,
                    "input_encoding", "AL32UTF8",
                    "no_ri_open",  "TRUE", /* disallow URI resolution */
                    "error_handler", test_errmsg,
                    NULL);
```

The parameter `no_ri_open` can also be set to `TRUE` in DOM APIs to block insecure accesses. Setting this flag in DOM APIs ensures that security is enforced even when this flag is not set in `XmlCreate()`.

For example:

```
doc = XmlLoadDom(xctx, &xerr,
                 "file", filename,
                 "discard_whitespace", TRUE,
                 "validate", TRUE,
                 "no_ri_open", TRUE, /* disallow URI resolution */
                 NULL);
```

Allow Arbitrary Accesses

When users want to allow arbitrary references to external resources or make the accesses under their own control, they can implement the following steps:

1. Set `no_ri_open` to `FALSE` in the locations shown in the previous examples.

Copyright © 1999, 2025, Oracle and/or its affiliates.

2. Develop customized open, read and close callback functions and initialize an `OraSteam` using these functions.
3. Override access to `xctx` with `OraSteam`

For example:

```
ostream = OraStreamInit(actx, (oratext *)"test", &oerr,
                        "open", http_open,
                        "read", http_read,
                        "close", http_close,
                        NULL);
xerr = XmlAccess(xctx, XML_ACCESS_HTTP, ostream);
```

The code creates an instance of `ostream` by providing three callbacks. The created stream will override the access inside `xctx` using `XmlAccess`. `xctx` is used when calling APIs such as `XmlLoadDom`. Hence, the user has the option to create rules based on which they can access external resources.

2.3 Security for C++

By default, C++ APIs block accesses to external resources – thereby ensuring security. However, users can override this security restriction and allow arbitrary references to external resources based on their discretion.

The `xmlctx.hpp` header file uses the `CXmlCtx(bool no_ri_open)` API to initialize the `xctx` context and set the parameter `no_ri_open`. When the `no_ri_open` parameter is set to `true`, parser APIs that use `xctx` will not be able to access external resources. When the `no_ri_open` parameter is set to `false`, parser APIs that use `xctx` will be able to access external resources.

By default, parser APIs that use `xctx` cannot access external resources. To allow access, set the `no_ri_open` parameter to `false` as shown below:

```
xctx = CXmlCtx(FALSE); /* allow URI resolution */
```

> ⚠ **Caution**
>
> Allowing access to external resources can cause XML External Entity (XXE) and XML Entity Expansion (XEE) attacks. The user should be cautions when overriding the security restriction and allowing access to external resources.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Part I
Oracle XML Developer's Kit for C

An explanation is given of how to use Oracle XML Developer's Kit (XDK) to develop C applications.

Copyright © 1999, 2025, Oracle and/or its affiliates.

3

Getting Started with Oracle XML Developer's Kit for C

An explanation is given of how to get started with Oracle XML Developer's Kit (XDK) for C.

3.1 Installing XDK for C Components

XDK for C components are the building blocks for reading, manipulating, transforming, and validating Extensible Markup Language (XML). The XDK for C components are included with Oracle Database.

> ⚠ **Caution**
>
> Using the components of Oracle XML Developer’s Kit (XDK) to build software programs enables some powerful but potentially dangerous features, such as external entity expansion and recursive expansion. Refer to Security Considerations for Oracle XML Developer's Kit for information about how to use XDK securely.

This chapter assumes that you have installed XDK with Oracle Database and also installed the demo programs on the Oracle Database Examples media. See About Installing XDK for installation instructions and a description of the XDK directory structure.

The following set of examples shows the UNIX directory structure for the XDK demos and the libraries used by the XDK components. The subdirectories contain sample programs and data files for the XDK for C components.

Example 3-1 lists the main directories under the Oracle home directory for C.

The contents of each subdirectory under this main directory are listed individually.

The `bin` directory contains these components:

```
schema
xml
xmlcg
xsl
xvm
```

The `lib` directory contains these components:

```
libcore21.a
libcoresh21.so
libnls21.a
libunls21.dll
libxml21.a
libxmlsh21.a
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

The `xdk` directory contains this `demo` subdirectory:

```
| demo/
   | - c/
        | - dom/
        | - parser/
        | - sax/
        | - schema/
        | - webdav/
        | - xslt/
        | - xsltvm/
```

The `/xdk/demo/c` subdirectories contain sample programs and data files for XDK for C components. The chapters in Oracle XML Developer's Kit for C explain how to use these programs to gain an understanding of the most important C features.

The `xdk` directory also contains this `include` subdirectory:

```
| include/
   oratypes.h
   oraxml.h
   oraxmlcg.h
   oraxsd.h
   xml.h
   xmlerr.h
   xmlotn.h
   xmlproc.h
   xmlsch.h
   xmlxptr.h
   xmlxsl.h
   xmlxvm.h
```

Table 3-4 in Setting Up and Testing the XDK C Compile-Time Environment on UNIX describes the C header files.

Example 3-1 Oracle XML Developer's Kit for C Libraries, Header Files, Utilities, and Demos

```
- $ORACLE_HOME
    | - bin/
    | - lib/
    | - xdk/
```

Related Topics

- Overview of XDK
  Oracle XML Developer’s Kit (XDK) is a versatile set of components that enables you to build and deploy C, C++, and Java software programs that process Extensible Markup Language (XML). You can assemble these components into an XML application that serves your business needs.

Copyright © 1999, 2025, Oracle and/or its affiliates.

3.2 Configuring the UNIX Environment for XDK for C Components

Topics here include component dependencies, environment variables, the runtime and compile-time environments, and the component version.

3.2.1 XDK for C Component Dependencies on UNIX

The C libraries described in this section are located in `$ORACLE_HOME/lib`.

XDK for C and C++ components are contained in this library:

`libxml21.a`

The following XKD components are contained in the library:

- XML parser, which checks an XML document for well-formedness, optionally validates it against a document type definition (DTD) or XML Schema, and supports Document Object Model (DOM) and Simple API for XML (SAX) interfaces for programmatic access
- Extensible Stylesheet Language Transformation (XSLT) processor, which transforms an XML document into another XML document
- XSLT compiler, which compiles XSLT stylesheets into byte code for use by the XSLT Virtual Machine (XSLT VM)
- XSLTVM, which is an XSLT transformation engine
- XML Schema processor, which validates XML files against an XML schema

Table 3-1 describes the Common Oracle Runtime Environment (CORE) and Globalization Support libraries on which XDK for C components (UNIX) depend.

Table 3-1 Dependent Libraries of Oracle XML Developer's Kit for C Components on UNIX

Component	Library	Description
CORE library	`libcore21.a`	Contains the C runtime functions that enable portability across platforms.
CORE Dynamic linking library	`libcoresh21.so`	C runtime library that supports dynamic linking on UNIX platforms.
Globalization Support common library	`libnls21.dll`	Supports the 8-bit encoding of Unicode (UTF-8), 16-bit encoding of Unicode (UTF-16), and ISO-8859-1 character sets. This library depends on the environment to locate encoding and message files.
Globalization Support library for Unicode	`libunls21.a`	Supports the character sets described in *Oracle AI Database Globalization Support Guide*. This library depends on the environment to locate encoding and message files.

Copyright © 1999, 2025, Oracle and/or its affiliates.

3.2.2 Setting Up XDK for C Environment Variables on UNIX

The UNIX environment variables required for use with XDK for C components is described.

Table 3-2 UNIX Environment Settings for Oracle XML Developer's Kit for C Components

Variable	Description	Setting
`$ORA_NLS10`	Sets the location of the Globalization Support character-encoding definition files. The encoding files represent a subset of character sets available in Oracle Database.	Set to the location of the Globalization Support data files. Set the variable: `setenv ORA_NLS10 $ORACLE_HOME/nls/data`
`$ORA_XML_MESG`	Sets the location of the XML error message files. Files ending in `.msb` are machine-readable and required at run time. Files ending in `.msg` are human-readable and contain cause and action descriptions for each error.	Set to the path of the `mesg` directory. For example: `setenv ORA_XML_MESG $ORACLE_HOME/xdk/mesg`
`$PATH`	Sets the location of the XDK for C executables.	Set the PATH: `setenv PATH ${PATH}:${ORACLE_HOME}/bin`

3.2.3 Testing the XDK for C Runtime Environment on UNIX

You can test XDK for C in your UNIX runtime environment by running a number of utilities.

These utilities are described in Table 3-3.

Table 3-3 Oracle XML Developer's Kit for C/C++ Utilities on UNIX

Executable	Directory	Description
`schema`	`$ORACLE_HOME/bin`	C XML Schema validator
`xml`	`$ORACLE_HOME/bin`	C XML parser
`xmlcg`	`$ORACLE_HOME/bin`	C++ class generator
`xvm`	`$ORACLE_HOME/bin`	C XVM processor

Run these utilities with no options to display the usage help. Run the utilities with the `-hh` flag for complete usage information.

Related Topics

- Using the C XML Schema Processor Command-Line Utility
  You can call XML Schema processor for C as an executable by invoking `bin/schema` in the install area.
- Using the C XML Parser Command-Line Utility
  The `xml` utility, which is located in `$ORACLE_HOME/bin` (UNIX) or `%ORACLE_HOME%\bin` (Windows), is a command-line interface that parses XML documents. It checks for both well-formedness and validity.
- Using the XML C++ Class Generator Command-Line Utility
  The standalone class generator can be called as an executable by invoking `bin/xmlcg`.

Copyright © 1999, 2025, Oracle and/or its affiliates.

- Using the XVM Processor Command-Line Utility
  The XVM processor is accessed from the command-line using command `xvm`.

3.2.4 Setting Up and Testing the XDK C Compile-Time Environment on UNIX

How to set up and test the XDK C compile-time UNIX environment is described.

Table 3-4 describes the header files required for compilation of XDK for C components. These files are located in `$ORACLE_HOME/xdk/include`. Your runtime environment must be set up before you can compile your code.

Table 3-4 Header Files in the Oracle XML Developer's Kit for C Compile-Time Environment

Header File	Description
`oratypes.h`	Includes the private Oracle C data types.
`oraxml.h`	Includes the Oracle9i XML Open Reporting Application (ORA) data types and the public ORA APIs included in `libxml.a` (only for backward compatibility). Use `xml.h` instead.
`oraxmlcg.h`	Includes the C APIs for the C++ class generator (only for backward compatibility).
`oraxsd.h`	Includes the Oracle9i XML schema definition (XSD) validator data types and application programming interfaces (APIs), for backward compatibility only.
`xml.h`	Handles the unified DOM APIs transparently, whether you use them through Oracle Call Interface (OCI) or standalone. It replaces `oraxml.h`, which is deprecated.
`xmlerr.h`	Includes the XML errors and their numbers.
`xmlotn.h`	Includes the other headers depending on whether you compile standalone or use OCI.
`xmlproc.h`	Includes the Oracle XML data types and XML public parser APIs in `libxml21.a`.
`xmlsch.h`	Includes the Oracle XSD validator public APIs.
`xmlptr.h`	Includes the XPointer data types and APIs, which are not currently documented or supported.
`xmlxsl.h`	Includes the XSLT processor data types and public APIs.
`xmlxvm.h`	Includes the XSLT compiler and VM data types and public APIs.

3.2.4.1 Testing the XDK for C Compile-Time Environment on UNIX

The simplest way to test XDK for C in your compile-time environment is to run the `make` utility on the sample programs, which are located on the Examples media rather than on the Oracle Database CD.

After installing the demos, they are located in `$ORACLE_HOME/xdk/demo/c`. A `README` in the same directory provides compilation instructions and usage notes.

Build and run the sample programs by executing these commands at the system prompt:

```
cd $ORACLE_HOME/xdk/demo/c
make
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

3.2.5 Verifying the XDK for C Component Version on UNIX

How to determine which version of XDK you have is explained.

To get the version of XDK you are working with, change to directory `$ORACLE_HOME/lib` and run this command:

```
strings libxml21.a | grep -i developers
```

3.3 Configuring the Windows Environment for XDK C Components

Topics here include component dependencies, setting environment variables, testing the runtime environment, setting up and testing the compile-time environment, and Visual C++ in Microsoft Visual Studio.

3.3.1 XDK for C Component Dependencies on Windows

The C libraries described in this section are located in `%ORACLE_HOME%\bin`.

XDK for C components are contained in this library:

```
libxml21.dll
```

The following XDK components are contained in the library:

- XML parser
- XSLT processor
- XSLT compiler
- XSLT VM
- XML Schema processor

Table 3-5 describes the Oracle CORE and Globalization Support libraries on which XDK for C components (Windows) depend.

Table 3-5 Dependent Libraries of Oracle XML Developer's Kit for C Components on Windows

Component	Library	Description
CORE library	`libcore21.dll`	Contains the runtime functions that enable portability across platforms.
Globalization Support common library	`libnls21.dll`	Supports the UTF-8, UTF-16, and ISO-8859-1 character sets. This library depends on the environment to find encoding and message files.
Globalization Support library for Unicode	`libunls21.dll`	Supports the character sets described in *Oracle AI Database Globalization Support Guide*. This library depends on the environment to find encoding and message files.

Copyright © 1999, 2025, Oracle and/or its affiliates.

3.3.2 Setting Up XDK for C Environment Variables on Windows

The Windows environment variables required for use with the XDK for C components are described.

Table 3-6 Windows Environment Settings for Oracle XML Developer's Kit for C Components

Variable	Description	Setting
%ORA_NLS10%	Sets the location of the Globalization Support character-encoding definition files. The encoding files represent a subset of character sets available in Oracle Database.	This variable must be set to the location of the Globalization Support data files. Set the variable: `set ORA_NLS10=%ORACLE_HOME%\nls\data`
%ORA_XML_MESG%	Sets the location of the XML error message files. Files ending in `.msb` are machine-readable and required at run time. Files ending in `.msg` are human-readable and contain cause and action descriptions for each error.	Set to the path of the `mesg` directory. For example: `set ORA_XML_MESG=%ORACLE_HOME%\xdk\mesg`
%PATH%	Sets the location of the XDK for C data definition languages (DLLs) and executables.	Set the PATH: `path %path%;%ORACLE_HOME%\bin`

3.3.3 Testing the XDK for C Runtime Environment on Windows

You can test XDK in your Microsoft Windows runtime environment by running a number of utilities.

These are described in Table 3-7.

Table 3-7 Oracle XML Developer's Kit for C/C++ Utilities on Windows

Executable	Directory	Description
`schema.exe`	`%ORACLE_HOME%\bin`	C XML Schema validator See also Using the C XML Schema Processor Command-Line Utility
`xml.exe`	`%ORACLE_HOME%\bin`	C XML parser See also Using the C XML Parser Command-Line Utility
`xmlcg.exe`	`%ORACLE_HOME%\bin`	C++ class generator See also Using the XML C++ Class Generator Command-Line Utility
`xvm.exe`	`%ORACLE_HOME%\bin`	C XVM processor See also Using the XVM Processor Command-Line Utility

Run these utilities with no options to display the usage help. Run the utilities with the `-hh` flag for complete usage information.

Related Topics

- Using the C XML Schema Processor Command-Line Utility
  You can call XML Schema processor for C as an executable by invoking `bin/schema` in the install area.

Copyright © 1999, 2025, Oracle and/or its affiliates.

- Using the C XML Parser Command-Line Utility
  The `xml` utility, which is located in `$ORACLE_HOME/bin` (UNIX) or `%ORACLE_HOME%\bin` (Windows), is a command-line interface that parses XML documents. It checks for both well-formedness and validity.
- Using the XML C++ Class Generator Command-Line Utility
  The standalone class generator can be called as an executable by invoking `bin/xmlcg`.
- Using the XVM Processor Command-Line Utility
  The XVM processor is accessed from the command-line using command `xvm`.

3.3.4 Setting Up and Testing the XDK for C Compile-Time Environment on Windows

You must set up your runtime environment before you can compile your code.

Table 3-4 in the section Setting Up and Testing the XDK C Compile-Time Environment on UNIX describes the header files required for compilation of the C components on Windows. The relative file names are the same on both UNIX and Windows installations.

On Windows the header files are located in `%ORACLE_HOME%\xdk\include`.

3.3.4.1 Testing the XDK for C Compile-Time Environment on Windows

You can test XDK for C in your compile-time environment by compiling the demo programs.

These are located in `%ORACLE_HOME%\xdk\demo\c` after you install them from the Oracle Database Examples media. A `README` file in the same directory provides compilation instructions and usage notes. Before you compile the demo programs, edit the `Make.bat` files as described in Editing the Make.bat Files on Windows.

3.3.4.1.1 Editing the Make.bat Files on Windows

Each subfolder of folder `%ORACLE_HOME%\xdk\demo\c` contains a file `Make.bat`. Update the `Make.bat` file in each folder by adding the path of the libraries and the header files to the compile command. You need not edit the paths in section `:LINK` because `/libpath:%ORACLE_HOME%\lib` already points to the C libraries.

The section of a `Make.bat` file in Example 3-2 uses bold text to show the path that you must include.

Example 3-2 Editing an Oracle XML Developer's Kit for C Make.bat File on Windows

```
: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
/Doratext=OraText -I. -I..\..\..\include -I%ORACLE_HOME%\xdk\include %filename%.c
goto :EOF

:LINK
set filename=%1
link %link_dbg% /out:..\..\..\..\bin\%filename%.exe
/libpath:%ORACLE_HOME%\lib /libpath:..\..\..\..\lib
%filename%.obj oraxml21.lib user32.lib kernel32.lib msvcrt.lib ADVAPI32.lib
oldnames.lib winmm.lib
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

3.3.4.1.2 Setting the XDK for C Compiler Path on Windows

How to set the path for the `cl.exe` compiler on Microsoft Windows is described.

Demo file `make.bat` assumes that you are using the `cl.exe` compiler, which is freely available with the Microsoft .NET Framework Software Development Kit (SDK).

To set the path for the `cl.exe` compiler on Microsoft Windows, follow these steps:

1. In the **Start** menu, select **Settings** and then **Control Panel**.
2. Double-click **System**.
3. In the **System Properties** dialogue box, select the **Advanced** tab and click **Environment Variables**.
4. In **System variables**, select **Path** and click **Edit**.
5. Append the path of `cl.exe` to the `%PATH%` variable and click **OK**.

Build and run the sample programs by executing these commands at the system prompt:

```
cd $ORACLE_HOME/xdk/demo/c
make
```

3.3.5 Using the XDK for C Components and Visual C++ in Microsoft Visual Studio

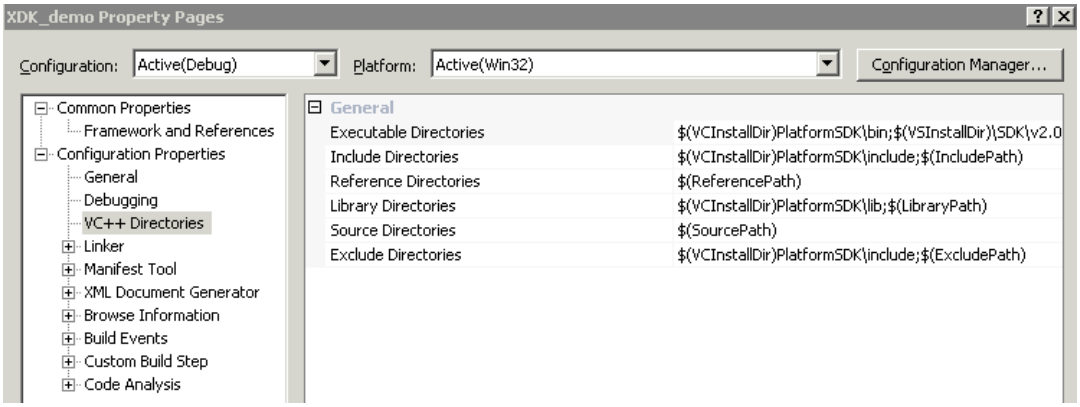
You can set up a project with a Visual C++ template and use it for the demos included in XDK.

3.3.5.1 Setting a Path for a Project in Visual C++ on Windows

Follow these steps to set the path for a project:

1. Open a project in Visual C++ and include the `*.c` files for your project.
2. Navigate to the **Project** menu and select **Properties**.
3. When Property Pages appear, expand **Configuration Properties** and select **VC++ Directories**.
4. Under General on the right side, select **Include Directories**.

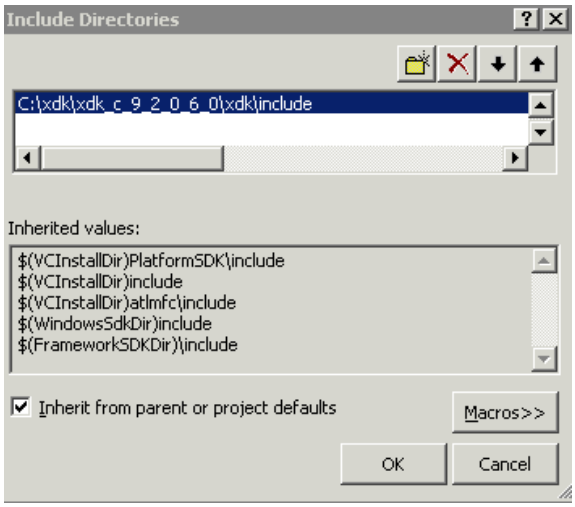
Figure 3-1 The Property Pages



Copyright © 1999, 2025, Oracle and/or its affiliates.

5. Click the arrow at the end of the line, and select the second line, which reads <Edit...>.
6. When the Include Directories window appears, click **New Line** from the tool bar and enter this include path, `%ORACLE_HOME%\xdk\include`, as shown in the example in Figure 3-2 and click **OK**.

Figure 3-2 Setting the Include Path in Visual C++



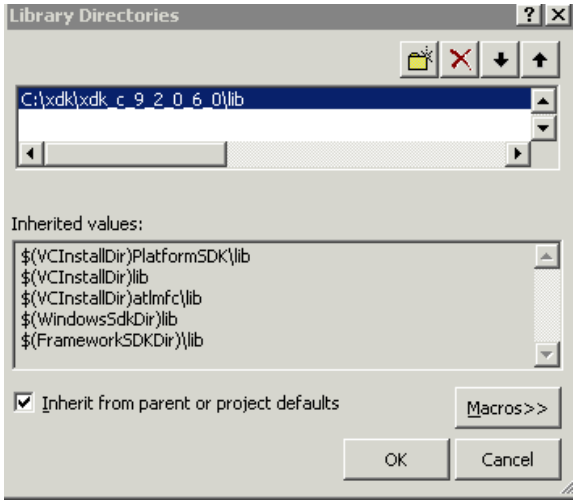
3.3.5.2 Setting the Library Path in Visual C++ on Windows

Follow these steps to set the library path for a project:

1. Open a project in Visual C++ and include the `*.c` files for your project.
2. Navigate to the **Project** menu and select **Properties**.
3. When Property Pages appear, expand **Configuration Properties** and select **VC++ Directories**.
4. Under General on the right side, select **Library Directories**.
5. Click the arrow at the end of the line, and select the second line which reads <Edit...>.
6. When the Library Directories window appears, click **New Line** from the tool bar and enter this library path, `%ORACLE_HOME%\lib`, as shown in the example in Figure 3-3 and click **OK**.

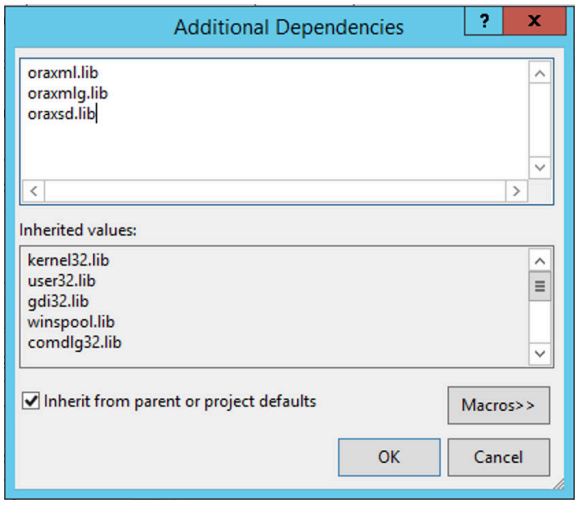
Copyright © 1999, 2025, Oracle and/or its affiliates.

Figure 3-3 Setting the Static Library Path in Visual C++



7. After setting the paths for the static libraries in `%ORACLE_HOME%\lib`, navigate to the **Project** menu and select **Properties**.
8. In the Properties Page, select and expand **Linker** under **Configuration Properties**, and select **Input**.
9. Select **Additional Dependencies** and click the arrow at the end of the line. Select the second line which reads <Edit...>.
10. Enter these additional dependencies: `oraxml21.lib`, `oraxmlg21.lib`, and `oraxsd21.lib` as shown in Figure 3-4 and click **OK**.

Figure 3-4 Setting the Names of the Libraries in Visual C++ Project



3.4 Overview of the Unified C API

The **unified C API** is a programming interface that unifies the functionality required by both XDK for C and Oracle XML DB. This API is used primarily by XSLT and XML Schema.

As shown in Table 3-4, the unified C API is declared in the `xml.h` header file. Table 3-8 summarizes the XDK for C APIs. See *Oracle AI Database XML C API Reference* for complete documentation.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 3-8 Summary of Oracle XML Developer's Kit for C APIs

Package	Purpose
Callback APIs	Define macros that declare functions (or function pointers) for XML callbacks.
DOM APIs	Parse and manipulate XML documents with DOM. The API follows the DOM 2.0 standard as closely as possible, although it changes some names when mapping from the objected-oriented DOM specification to the flat C namespace. For example, the overloaded `getName()` methods become `getAttrName()`.
Range APIs	Create and manipulate `Range` objects.
SAX APIs	Enable event-based XML parsing with SAX.
Schema APIs	Assemble multiple XML schema documents into a single schema that can be used to validate a specific instance document.
Traversal APIs	Enable document traversal and navigation of DOM trees.
XML APIs	Define an XML processor in terms of how it must read XML data and the information it must provide to the application.
XPath APIs	Process XPath-related types and interfaces.
XPointer APIs	Locate nodes in an XML document.
XSLT APIs	Perform XSL processing.
XSLTVM APIs	Implement a virtual machine that can run compiled XSLT code.

The API accomplishes the unification of the functions by conforming contexts. A top-level XML context (`xmlctx`) shares common information between cooperating XML components. This context defines information about:

- Data encoding
- Error message language
- Low-level allocation callbacks

An application needs this information before it can parse a document and provide programmatic access through DOM or SAX interfaces.

Both XDK for C and Oracle XML DB require different startup and tear-down functions for the top-level and service contexts. The initialization function takes implementation-specific arguments and returns a conforming context.

The unification is made possible by using conforming contexts. A conforming context means that the returned context must begin with a `xmlctx`; it may have any additional implementation-specific parts after the standard header.

After an application gets `xmlctx`, it uses unified DOM invocations, all of which take an `xmlctx` as the first argument.

3.5 Globalization Support for the XDK for C Components

The XDK for C parser supports over 300 IANA character sets.

These character sets include those listed in Character Sets Supported by XDK for C.

Considerations when working with character sets:

- Oracle recommends that you use Internet Assigned Numbers Authority (IANA) character set names for interoperability with other XML parsers.

Copyright © 1999, 2025, Oracle and/or its affiliates.

- XML parsers are required only to support UTF-8 and UTF-16, so these character sets are preferable.
- The default input encoding ("incoding") is UTF-8. If an input document's encoding is not self-evident (by HTTP character set, Byte Order Mark (BOM), XMLDecl, and so on), then the default input encoding is assumed. Oracle recommends 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 because single-byte performance is fastest. The flag `XML_FLAG_FORCE_INCODING` specifies that the default input encoding is always applied to input documents, ignoring any BOM or XMLDecl. Nevertheless, a protocol declaration such as HTTP character set is always honored.
- Choose the data encoding for DOM and SAX ("outcoding") carefully. Single-byte encodings are the fastest, but can represent only a very limited set of characters. Next fastest is Unicode (UTF-16), and slowest are the multibyte encodings such as UTF-8. If input data cannot be converted to the outcoding without loss, then an error occurs. For maximum utility, use a Unicode-based outcoding because Unicode can represent any character. If outcoding is not specified, then it defaults to the incoding of the first document parsed.

Copyright © 1999, 2025, Oracle and/or its affiliates.

4

Using the XSLT and XVM Processors for C

An explanation is given of how to use the Extensible Stylesheet Language Transformation (XSLT) and XSLT Virtual Machine (XVM) processors for C.

> **Note**
>
> Use the unified C application programming interface (API) for Oracle XML Developer's Kit (XDK) and Oracle XML DB applications. Older, nonunified C functions are deprecated and supported only for backward compatibility. They will be removed in a future release.
>
> The unified C API is described in Using the XML Parser for C.

4.1 XSLT XVM Processor

The Oracle XVM package includes the XSLT compiler and the XVM. This package implements the XSLT language as specified in the World Wide Web Consortium (W3C) Recommendation of 16 November 1999.

Implementing the XSLT compiler and the XVM enables compilation of XSLT (Version 1.0) into bytecode format, which is executed by the virtual machine. XVM is the software implementation of a CPU designed to run compiled XSLT code. The virtual machine assumes a compiler compiling XSLT stylesheets to a sequence of bytecodes or machine instructions for the XSLT CPU. The bytecode program is a platform-independent sequence of 2-byte units. It can be stored, cached, and run on different XVMs. The XVM uses the bytecode programs to transform XML instance documents. This approach clearly separates compile-time from runtime computations and specifies a uniform way of exchanging data between instructions.

The benefits of this approach are:

- An XSLT stylesheet can be compiled, saved in a file, and reused often, even on different platforms.
- The XVM is significantly faster and uses less memory than other XSLT processors.
- The bytecodes are language independent. There is no difference between code generated from a C or C++ XSLT compiler.

4.1.1 XVM Usage Example

A typical scenario of using the package APIs is described.

1. Create and use an XML meta-context object.

   ```
   xctx = XmlCreate(&err,...);
   ```

2. Create and use an XSLT compiler object.

   ```
   comp = XmlXvmCreateComp(xctx);
   ```


Copyright © 1999, 2025, Oracle and/or its affiliates.

3. Compile an XSLT stylesheet or XPath expression and store or cache the resulting bytecode.

```
code = XmlXvmCompileFile(comp, xslFile, baseuri, flags, &err);
```

or

```
code = XmlXvmCompileDom (comp, xslDomdoc, flags, &err);
```

or

```
code = XmlXvmCompileXPath (comp, xpathexp,  namespaces, &err);
```

4. Create and use an XVM object. The explicit stack size setting is needed when XVM terminates with a `Stack Overflow` message or when smaller memory footprints are required. See `XmlXvmCreate()`.

```
vm = XmlXvmCreate(xctx, "StringStack", 32, "NodeStack", 24, NULL);
```

5. Set the output (optional). Default is a stream.

```
err = XmlXvmSetOutputDom (vm, NULL);
```

or

```
err = XmlXvmSetOutputStream(vm, &xvm_stream);
```

or

```
err = XmlXvmSetOutputSax(vm, &xvm_callback, NULL);
```

6. Set a stylesheet bytecode to the XVM object. Can be repeated with other bytecode.

```
len = XmlXvmGetBytecodeLength(code, &err);
err = XmlXvmSetBytecodeBuffer(vm, code, len);
```

or

```
err = XmlXvmSetBytecodeFile (vm, xslBytecodeFile);
```

7. Transform an instance XML document or evaluate a compiled XPath expression. Can be repeated with the same or other XML documents.

```
err = XmlXvmTransformFile(vm, xmlFile, baseuri);
```

or

```
err = XmlXvmTransformDom (vm, xmlDomdoc);
```

or

```
obj  = (xvmobj*)XmlXvmEvaluateXPath (vm, code, 1, 1, node);
```

8. Get the output tree fragment (if DOM output is set at Step 5).

```
node = XmlXvmGetOutputDom (vm);
```

9. Delete the objects.

```
XmlXvmDestroy(vm);
XmlXvmDestroyComp(comp);
XmlDestroy(xctx);
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

4.1.2 Using the XVM Processor Command-Line Utility

The XVM processor is accessed from the command-line using command `xvm`.

```
xvm
```

Usage:

```
xvm options xslfile xmlfile

xvm options xpath xmlfile
```

Options:

```
-c        Compile xslfile. The bytecode is in "xmlfile.xvm".
-ct       Compile xslfile and transform xmlfile.
-t        Transform xmlfile using bytecode from xslfile.
-xc       Compile xpath. The bytecode is in "code.xvm".
-xct      Compile and evaluate xpath with xmlfile.
-xt       Evaluate XPath bytecode from xpath with xmlfile.
```

Examples:

```
xvm -ct  db.xsl db.xml
xvm -t   db.xvm db.xml
xvm -xct "doc/employee[15]/family" db.xml
```

4.1.3 Accessing the XVM Processor for C

Oracle XVM Processor for C is part of the standard installation of Oracle Database.

> **See Also**
>
> - *Oracle AI Database XML C API Reference*, XSLTVM APIs for C
> - XDK on OTN

4.2 XSLT Processor for XDK for C

The Oracle XSL/XPath package implements the XSLT language as specified in the W3C Recommendation of 16 November 1999. The package includes the XSLT 1.0 processor and XPath 1.0 Processor. The Oracle implementation of the XSLT processor follows the common design approach of melding compiler and processor into one object.

> **See Also**
>
> - *XSL Transformations (XSLT)*
> - *XML Path Language (XPath)*

4.2.1 XSLT Processor Usage Example

A typical scenario of using the package APIs is presented.

Copyright © 1999, 2025, Oracle and/or its affiliates.

1. Create and use an XML meta-context object.

```
xctx = XmlCreate(&err,...);
```

2. Parse the XSLT stylesheet.

```
xslDomdoc = XmlLoadDom(xctx, &err, "file", xslFile, "base_uri", baseuri,
NULL);
```

3. Create an XSLT processor for the stylesheet

```
xslproc = XmlXslCreate (xctx, xslDomdoc, baseuri, &err);
```

4. Parse the instance XML document.

```
xmlDomdoc = XmlLoadDom(xctx, &err, "file", xmlFile,  "base_uri", baseuri,
NULL);
```

5. Set the output (optional). Default is Document Object Model (DOM).

```
err = XmlXslSetOutputStream(xslproc, &stream);
```

6. Transform the XML document. This step can be repeated with the same or other XML documents.

```
err = XmlXslProcess (xslproc, xmlDomdoc, FALSE);
```

7. Get the output (if DOM).

```
node = XmlXslGetOutput(xslproc);
```

8. Delete objects.

```
XmlXslDestroy(xslproc);
XmlDestroy(xctx);
```

4.2.2 XPath Processor Usage Example

A typical scenario of using the package APIs is described.

Follow these steps:

1. Create and use an XML meta-context object.

```
xctx = XmlCreate(&err,...);
```

2. Parse the XML document or get the current node from already existing DOM.

```
node = XmlLoadDom(xctx, &err, "file", xmlFile,  "base_uri", baseuri, NULL);
```

3. Create an XPath processor.

```
xptproc = XmlXPathCreateCtx(xctx, NULL, node, 0, NULL);
```

4. Parse the XPath expression.

```
exp = XmlXPathParse (xptproc, xpathexpr, &err);
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

5. Evaluate the XPath expression.

```
obj = XmlXPathEval(xptproc, exp, &err);
```

6. Delete the objects.

```
XmlXPathDestroyCtx (xptproc);
XmlDestroy(xctx);
```

4.2.3 Using the C XSLT Processor Command-Line Utility

You can call the C Oracle XSLT processor as an executable by invoking `bin/xsl`.

```
xsl [switches] stylesheet instance
or
xsl -f [switches] [document filespec]
```

If no stylesheet is provided, no output is generated. If there is a stylesheet, but no output file, output goes to `stdout`.

Table 4-1 lists the command line options.

Table 4-1 XSLT Processor for C: Command Line Options

Option	Description
`-B BaseUri`	Set the Base URI for XSLT processor: `BaseUri` of `http://pqr/xsl.txt` resolves `pqr.txt` to `http://pqr/pqr.txt`
`-e encoding`	Specify default input file encoding (`-ee` to force).
`-E encoding`	Specify DOM or Simple API for XML (SAX) encoding.
`-f`	File—interpret as filespec, not Universal Resource Identifier (URI).
`-G xptrexprs`	Evaluates XPointer schema examples given in a file.
`-h`	Help—show this usage. (Use `-hh` for more options.)
`-hh`	Show complete options list.
`-i n`	Number of times to iterate the XSLT processing.
`-l language`	Language for error reporting.
`-o XSLoutfile`	Specifies output file of XSLT processor.
`-v`	Version—display parser version then exit.
`-V var value`	Test top-level variables in C XSLT.
`-w`	White Space—preserve all white space.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 4-1 (Cont.) XSLT Processor for C: Command Line Options

Option	Description
`-W`	Warning—stop parsing after a warning.

4.2.4 Accessing Oracle XSLT processor for C

Oracle XSLT processor for C is part of the standard installation of Oracle Database.

> **See Also**
>
> - *Oracle AI Database XML C API Reference*, XSLT APIs for C
> - *Oracle AI Database XML C API Reference*, XPath APIs for C
> - XDK on OTN

4.3 Using the Demo Files Included with the Software

Directory `$ORACLE_HOME/xdk/demo/c/parser/` contains several XML applications that show how to use the XSLT for C.

Table 4-2 XSLT for C Demo Files

Sample File Name	Description
`XSLSample.c`	Source for `XSLSample` program.
`XSLSample.std`	Expected output from `XSLSample`.
`class.xml`	XML file that can be used with `XSLSample`.
`iden.xsl`	Stylesheet that can be used with `XSLSample`.
`cleo.xml`	XML version of Shakespeare's play.
`XVMSample.c`	Sample usage of XVM and compiler. It takes two file names as input—XML file and XSLT stylesheet file.
`XVMXPathSample.c`	Sample usage of XVM and compiler. It takes XML file name and `XPath` expression as input. Generates the result of the evaluated `XPath` expression.
`XSLXPathSample.c`	Sample usage of `XSL/XPath` processor. It takes XML file name and `XPath` expression as input. Generates the result of the evaluated `XPath` expression.

4.3.1 Building the C Demo Programs for XSLT

Change directories to the demo directory and read the `README` file. That file explains how to build the sample programs according to your operating system.

Here is the usage of XSLT processor sample `XSLSample`, which takes two files as input, the XML file and the XSLT stylesheet:

Copyright © 1999, 2025, Oracle and/or its affiliates.

```
XSLSample xmlfile xslss
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

5

Using the XML Parser for C

An explanation is given of how to use the Extensible Markup Language (XML) parser for C.

5.1 Introduction to the XML Parser for C

Topics here include prerequisites and standards for the XML parser for C.

> **See Also**
>
> Introduction to XML Parsing for Java for a generic introduction to XML parsing with Document Object Model (DOM) and Simple API for XML (SAX). Much of the information in the introduction is language-independent and applies equally to C.

5.1.1 Prerequisites for Using the XML Parser for C

The Oracle XML parser for C reads an XML document and uses DOM or SAX application programming interfaces (APIs) to provide programmatic access to its content and structure. You can use the parser in validating or nonvalidating mode. A pull parser is also available.

This chapter assumes that you are familiar with these technologies:

- Document Object Model (DOM). DOM is an in-memory tree representation of the structure of an XML document.
- Simple API for XML (SAX). SAX is a standard for event-based XML parsing.
- Using the XML Pull Parser for C. Pull Parser uses XML events.
- document type definition (DTD). An XML DTD defines the legal structure of an XML document.
- XML Schema. Like a DTD, an XML schema defines the legal structure of an XML document.
- XML Namespaces. Namespaces are a mechanism for differentiating element and attribute names.

If you require a general introduction to the preceding technologies, consult the XML resources listed in Related Documents.

5.1.2 Standards and Specifications for the XML Parser for C

The standards and specifications for the XDK XML parser are described.

- XML 1.0 is a W3C Recommendation. The Oracle XML Developer's Kit (XDK) for C API provides full support for XML 1.0 (Second Edition).

Copyright © 1999, 2025, Oracle and/or its affiliates.

- The DOM Level 1, Level 2, and Level 3 specifications are World Wide Web Consortium (W3C) Recommendations. The XDK for C API provides full support for DOM Level 1 and 2, but no support for Level 3.
- SAX is available in version 1.0, which is deprecated, and 2.0. SAX is not a W3C specification. The XDK for C API provides full support for both SAX 1.0 and 2.0.
- XML Namespaces is a W3C Recommendation.

Related Topics

- Oracle XML Developer's Kit Standards
  A description is given of the Oracle XML Developer's Kit (XDK) standards.

5.2 Using the XML Parser API for C

Oracle XML parser for C checks if an XML document is well-formed, and optionally validates it against a DTD. Your application can access the parsed data through the DOM or SAX APIs.

5.2.1 Overview of the Parser API for C

The core of the XML parsing API are the XML, DOM, and SAX APIs.

Table 5-1 describes the interfaces for these APIs. See *Oracle AI Database XML C API Reference* for the complete API documentation.

Table 5-1 Interfaces for XML, DOM, and SAX APIs

Package	Interfaces	Function Name Convention
XML	This package implements a single `XML` interface. The interface defines functions for these tasks: • Creating and destroying contexts. A top-level XML context (`xmlctx`) shares common information between cooperating XML components. • Creating and parsing XML documents and DTDs.	Function names begin with the string `Xml`. See *Oracle AI Database XML C API Reference* for API documentation.
DOM	This package provides programmatic access to parsed XML. The package implements these interfaces: • `Attr` defines get and set functions for XML attributes. • `CharacterData` defines functions for manipulating character data. • `Document` defines functions for creating XML nodes, getting information about an XML document, and setting the DTD for a document. • `DocumentType` defines get functions for DTDs. • `Element` defines get and set functions for XML elements. • `Entity` defines get functions for XML entities. • `NamedNodeMap` defines get functions for named nodes. • `Node` defines get and set functions for XML nodes. • `NodeList` defines functions that free a node list and get a node from a list. • `Notation` defines functions that get the system and public ID from a node. • `ProcessingInstruction` defines get and set functions for processing instructions. • `Text` defines a function that splits a text node into two.	Function names begin with the string `XmlDom`. See *Oracle AI Database XML C API Reference* for API documentation.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 5-1 (Cont.) Interfaces for XML, DOM, and SAX APIs

Package	Interfaces	Function Name Convention
SAX	This package provides programmatic access to parsed XML. The package implements the SAX interface, which defines functions that receive notifications for SAX events.	Function names begin with the string `XmlSax`. See *Oracle AI Database XML C API Reference* for API documentation.
XML Pull Parser	XML events is a representation of an XML document which is similar to SAX events in that the document is represented as a sequence of events like start tag, end tag, comment, and so on. The difference is that SAX events are driven by the parser (producer) and XML events are driven by the application (consumer).	Function names begin with the string `XmlEv`. See *Oracle AI Database XML C API Reference* for API documentation.

5.2.1.1 XML Parser for C Data Types

The data types used in the XML parser for C are described.

See *Oracle AI Database XML C API Reference* for the complete list of data types for XDK for C.

Table 5-2 Data Types Used in the XML Parser for C

Data Type	Description
`oratext`	String pointer
`xmlctx`	Master XML context
`xmlsaxcb`	SAX callback structure (SAX only)
`ub4`	32-bit (or larger) unsigned integer
`uword`	Native unsigned integer

5.2.1.2 XML Parser for C Defaults

The defaults for the XML parser for C are described.

These are the defaults:

- Character set encoding is 8-bit encoding of Unicode (UTF-8). If all your documents are ASCII, then setting the encoding to US-ASCII increases performance.
- The parser prints messages to `stderr` unless an error handler is provided.
- The parser checks inputs documents for well-formedness but not validity. You can set the property "validate" to validate the input.

> **Note**
>
> Oracle recommends 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 faster performance than is possible with multibyte character sets such as UTF-8.

Copyright © 1999, 2025, Oracle and/or its affiliates.

- The parser conforms with the XML 1.0 specification when processing white space, that is, the parser reports all white space to the application but indicates which white space can be ignored. However, some applications may prefer to set the property "discard-white space," which discards all white space between an end-element tag and this start-element tag.

See Also

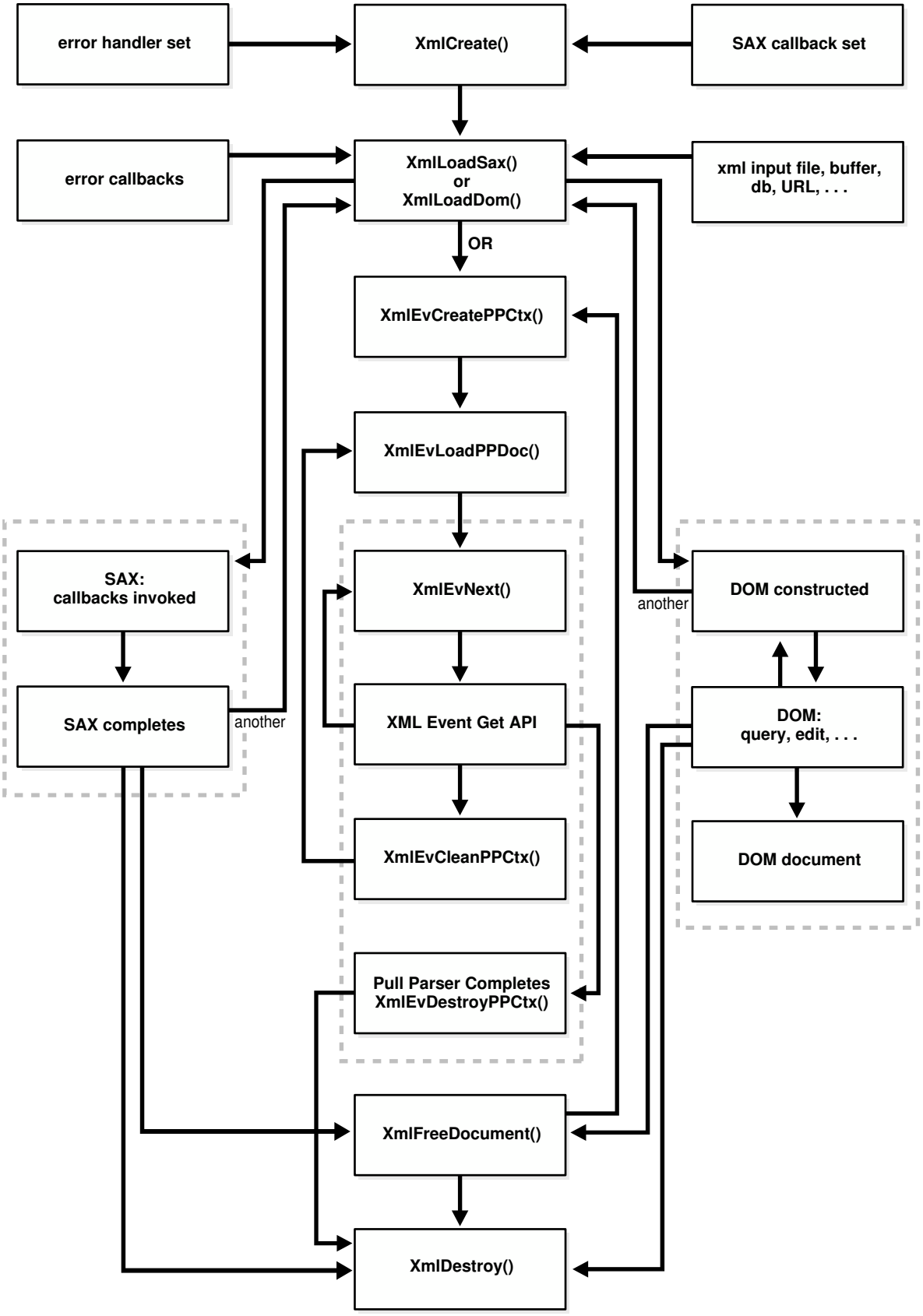
- *Oracle AI Database XML C API Reference* for the DOM, SAX, pull parser, and callback APIs.

5.2.2 XML Parser for C Calling Sequence

The calling sequence for the XML parser for C is illustrated.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Figure 5-1 XML Parser for C Calling Sequence



Copyright © 1999, 2025, Oracle and/or its affiliates.

5.2.3 Using the XML Parser for C: Basic Process

The basic process for using the XML parser for C is described.

Perform these steps in your application:

1. Initialize the parsing process with the `XmlCreate()` function. The following sample code fragment is from `DOMNamespace.c`:

   ```
   xmlctx     *xctx;
   ...
   xctx = XmlCreate(&ecode, (oratext *) "namespace_xctx", NULL);
   ```

2. Parse the input item, which can be an XML document or string buffer.

   If you are parsing with DOM, invoke the `XmlLoadDom()` function. The following sample code fragment is from `DOMNamespace.c`:

   ```
   xmldocnode *doc;
   ...
   doc = XmlLoadDom(xctx, &ecode, "file", DOCUMENT,
                    "validate", TRUE, "discard_whitespace", TRUE, NULL);
   ```

   If you are parsing with SAX, invoke the `XmlLoadSax()` function. The following sample code fragment is from `SAXNamespace.c`:

   ```
   xmlerr      ecode;
   ...
   ecode = XmlLoadSax(xctx, &sax_callback, &sc, "file", DOCUMENT,
                      "validate", TRUE, "discard_whitespace", TRUE, NULL);
   ```

   If you are using the pull parser, then include these steps to create the event context and load the document to parse:

   ```
   evctx = XmlEvCreatePPCtx(xctx, &xerr, NULL);
   XmlEvLoadPPDoc(xctx, evctx, "File", input_filenames[i], 0, NULL);
   ```

3. If you are using the DOM interface, then include these steps:

   - Use the `XmlLoadDom()` function to invoke `XmlDomGetDocElem()`. This step invokes other DOM functions, which are typically node or print functions that output the DOM document, as required. The following sample code fragment is from `DOMNamespace.c`:

     ```
     printElements(xctx, XmlDomGetDocElem(xctx, doc));
     ```

   - Invoke the `XmlFreeDocument()` function to clean up any data structures created during the parse process. The following sample code fragment is from `DOMNamespace.c`:

     ```
     XmlFreeDocument(xctx, doc);
     ```

   If you are using the SAX interface, then include these steps:

Copyright © 1999, 2025, Oracle and/or its affiliates.

- Process the results of the invocation of `XmlLoadSax()` with a callback function, such as:

```
xmlsaxcb saxcb = {
 UserStartDocument,  /* user's own callback functions */
 UserEndDocument,
 /* ... */
};

if (XmlLoadSax(xctx, &saxcb, NULL, "file", "some_file.xml", NULL) != 0)
  /* an error occured */
```

- Register the callback functions. You can set any of the SAX callback functions to `NULL` if not needed.

If you are using the pull parser, iterate over the events using:

```
cur_event = XmlEvNext(evctx);
```

Use the Get APIs to get information about that event.

4. Use `XmlFreeDocument()` to clean up the memory and structures used during a parse. The program does not free memory allocated for parameters passed to the SAX callbacks or for nodes and data stored with the DOM parse tree until you invoke `XMLFreeDocument()` or `XMLDestroy()`. The following sample code fragment is from `DOMNamespace.c`:

```
XmlFreeDocument(xctx, doc);
```

Either return to Step 2 or proceed to the next step.

For the pull parser invoke `XmlEvCleanPPCtx()` to release memory and structures used during the parse. The application can invoke `XmlEvLoadPPDoc()` again to parse another document. Or, it can invoke `XMLEvDestroyPPCtx()` after which the pull parser context cannot be used again.

```
XmlEvCleanPPCtx(xctx, evctx);
...
XmlEvDestroyPPCtx(xctx, evctx);
```

5. Terminate the parsing process with `XmlDestroy()`. The following sample code fragment is from `DOMNamespace.c`:

```
(void) XmlDestroy(xctx);
```

If threads fork off somewhere in the sequence of invocations between initialization and termination, the application produces unpredictable behavior and results.

You can use the memory callback functions `XML_ALLOC_F` and `XML_FREE_F` for your own memory allocation. If you do, then specify both functions.

Related Topics

- Using the XML Pull Parser for C
  The XML Pull Parser is an implementation of the XML Events interface. The XML Pull Parser and the SAX parser are similar, but using the Pull Parser, the application (consumer) drives the events, while in SAX, the parser (producer) drives the events.

Copyright © 1999, 2025, Oracle and/or its affiliates.

5.2.4 Running the XML Parser for C Demo Programs

The `$ORACLE_HOME/xdk/demo/c/` (UNIX) and `%ORACLE_HOME%\xdk\demo\c` (Windows) directories include several XML applications that show how to use the XML parser for C with the DOM and SAX interfaces.

Table 5-3 describes the demos.

The `make` utility compiles the source file *`fileName.c`* to produce the demo program *`fileName`* and the output file *`fileName.out`* . The *`fileName.std`* is the expected output.

Table 5-3 C Parser Demos

Directory	Contents	Demos

`dom`	`DOMNamespace.c`
	`DOMSample.c`
	`FullDom.c`
	`FullDom.xml`
	`NSExample.xml`
	`Traverse.c`
	`XPointer.c`
	`class.xml`
	`cleo.xml`
	`pantry.xml`
		The following demo programs use the DOM API:
		• The `DOMNamespace` program uses Namespace extensions to the DOM API. It prints out all elements and attributes of `NSExample.xml` along with full namespace information.
		• The `DOMSample` program uses DOM APIs to display an outline of *Cleopatra*, that is, the XML elements ACT and SCENE. The `cleo.xml` document contains the XML version of Shakespeare's *The Tragedy of Antony and Cleopatra*.
		• The `FullDom` program shows sample usage of the full DOM interface. It exercises all the invocations. The program accepts `FullDom.xml`, which shows the use of entities, as input.
		• The `Traverse` program shows the use of DOM iterators, tree walkers, and ranges. The program accepts the `class.xml` document, which describes a college Calculus course, as input.
		• The `XPointer` program shows the use of the XML Pointer Language by locating the children of the `<pantry>` element in `pantry.xml`.

`sax`	`NSExample.xml`
	`SAXNamespace.c`
	`SAXSample.c`
	`cleo.xml`
		The following demo programs use the SAX APIs:
		• The `SAXNamespace` program uses namespace extensions to the SAX API. It prints out all elements and attributes of `NSExample.xml` along with full namespace information.
		• The `SAXSample` program uses SAX APIs to show all lines in the play *Cleopatra* containing a given word. If you do not specify a word, then it uses the word "death." The `cleo.xml` document contains the XML version of Shakespeare's *The Tragedy of Antony and Cleopatra*.

You can find documentation that describes how to compile and run the sample programs in the `README` in the same directory. The basic steps are:

1. Change into the `$ORACLE_HOME/xdk/demo/c` directory (UNIX) or `%ORACLE_HOME%\xdk\demo\c` directory (Windows).
2. Make sure that your environment variables are set as described in Setting Up XDK for C Environment Variables on UNIX and Setting Up XDK for C Environment Variables on Windows.
3. Run `make` (UNIX) or `Make.bat` (Windows) at the system prompt. The `make` utility changes into each demo subdirectory and runs `make` to do this:
   a. Compiles the C source files with the `cc` utility. For example, the `Makefile` in the `$ORACLE_HOME/xdk/demo/c/dom` directory includes these line:

      ```
      $(CC) -o DOMSample $(INCLUDE) $@.c $(LIB)
      ```

   b. Runs each demo program and redirects the output to a file. For example, the `Makefile` in the `$ORACLE_HOME/xdk/demo/c/dom` directory includes this line:

Copyright © 1999, 2025, Oracle and/or its affiliates.

```
./DOMSample > DOMSample.out
```

4. Compare the `*.std` files to the `*.out` files for each program. The `*.std` file contains the expected output for each program. For example, `DOMSample.std` contains the expected output from running `DOMSample`.

5.2.5 Using the C XML Parser Command-Line Utility

The `xml` utility, which is located in `$ORACLE_HOME/bin` (UNIX) or `%ORACLE_HOME%\bin` (Windows), is a command-line interface that parses XML documents. It checks for both well-formedness and validity.

To use `xml` ensure that your environment is set up as described in Setting Up XDK for C Environment Variables on UNIX and Setting Up XDK for C Environment Variables on Windows.

Use this syntax on the command line to invoke `xml`. Use `xml.exe` for Windows:

```
xml [options] [document URI]
xml -f [options] [document filespec]
```

Table 5-4 describes the command-line options.

Table 5-4 C XML Parser Command-Line Options

Option	Description
`-B` BaseURI	Sets the base URI for the XSLT processor. The base URI of `http://pqr/xsl.txt` resolves `pqr.txt` to `http://pqr/pqr.txt`.
`-c`	Checks well-formedness, but performs no validation.
`-e` *`encoding`*	Specifies default input file encoding ("incoding").
`-E` *`encoding`*	Specifies DOM/SAX encoding ("outcoding").
`-f` *`file`*	Interprets the file as filespec, not URI.
`-G` *`xptr_exprs`*	Evaluates `XPointer` scheme examples given in a file.
`-h`	Shows usage help and basic list of command-line options.
`-hh`	Shows complete list command-line options.
`-i` *`n`*	Specifies the number of times to iterate the XSLT processing.
`-l` *`language`*	Specifies the language for error reporting.

`-n`	Traverses the DOM and reports the number of elements, as shown in this sample output:
	`ELEMENT 1`
	`PCDATA 1`
	`DOC 1`
	`TOTAL 3 * 60 = 180`

`-o` *`XSLoutfile`*	Specifies the output file of the XSLT processor.

`-p`	Prints the document/DTD structures after the parse. For example, the root element `<greeting>hello</greeting>` is printed as:
	`+---ELEMENT greeting`
	`+---PCDATA "hello"`

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 5-4 (Cont.) C XML Parser Command-Line Options

Option	Description
`-P`	Prints the document from the root element. For example, the root element `<greeting>hello</greeting>` is printed as: `<greeting>hello</greeting>`
`-PP`	Prints from the root node (DOC) and includes the XML declaration.
`-PE encoding`	Specifies the encoding for `-P` or `-PP` output.
`-PX`	Includes the XML declaration in the output.
`-s stylesheet`	Specifies the XSLT stylesheet.
`-v`	Displays the XDK parser version, and then exits.
`-V var value`	Tests top-level variables in CXSLT.
`-w`	Preserves all white space.
`-W`	Stops parsing after a warning.

`-x`	Exercises the SAX interface and prints the document, as shown in this sample output:
	`StartDocument`
	`XMLDECL version='1.0' encoding=FALSE`
	`<greeting>`
	`"hello"`
	`</greeting>`
	`EndDocument`

5.2.5.1 Using the XML Parser Command-Line Utility: Example

You can test `xml` documents using the various XML files located in `$ORACLE_HOME/xdk/demo/c`.

Example 5-1 displays the contents of `NSExample.xml`.

You can parse this file, count the number of elements, and display the DOM tree as shown in this example:

```
xml -np NSEample.xml > xml.out
```

Example 5-2shows the output.

Example 5-1 NSExample.xml

```
<!DOCTYPE doc [
<!ELEMENT doc (child*)>
<!ATTLIST doc xmlns:nsprefix CDATA #IMPLIED>
<!ATTLIST doc xmlns CDATA #IMPLIED>
<!ATTLIST doc nsprefix:a1 CDATA #IMPLIED>
<!ELEMENT child (#PCDATA)>
]>
<doc nsprefix:a1 = "v1" xmlns="http://www.w3c.org"
     xmlns:nsprefix="http://www.oracle.com">
<child>
This element inherits the default Namespace of doc.
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
</child>
</doc>
```

Example 5-2 xml.out

```
   ELEMENT       2
    PCDATA       1
       DOC       1
       DTD       1
  ELEMDECL       2
  ATTRDECL       3
     TOTAL      10 * 112 = 1120
+---ELEMENT doc [nsprefix:a1='v1'*, xmlns='http://www.w3c.org'*,
xmlns:nsprefix=
'http://www.oracle.com'*]
    +---ELEMENT child
        +---PCDATA "
This element inherits the default Namespace of doc.
"
```

5.3 Using the DOM API for C

Topics here include controlling encoding for XML documents, using NULL-terminated and length-encoded functions, and handling errors.

5.3.1 Controlling the Data Encoding of XML Documents for the C API

XML data occurs in many encodings. You can control the XML encoding in various ways.

- Specify a default encoding to assume for files that are not self-describing
- Specify the presentation encoding for DOM or SAX
- Re-encode when a DOM is serialized

Input XML data is always encoded. Some encodings are entirely self-describing, such as 16-bit encoding of Unicode (UTF-16), which requires a specific Byte Order Mark (BOM) before the start of the actual data. The `XMLDecl` or Multipurpose Internet Mail Extensions (MIME) header of the document can also specify an encoding. If the application cannot determine the specific encoding, then it applies the default input encoding. If you do not provide a default, then the application assumes UTF-8 on ASCII platforms and UTF-EBCDIC on EBCDIC platforms.

The API makes a provision for cases when the encoding data of the input document is corrupt. For example, suppose an ASCII document with an `XMLDecl` of `encoding=ascii` is blindly converted to EBCDIC. The new EBCDIC document contains (in EBCDIC) an `XMLDecl` that incorrectly claims the document is ASCII. The correct behavior for a program that is re-encoding XML data is to regenerate but not convert the `XMLDecl`. The `XMLDecl` is metadata, not data itself. This rule is often ignored, however, which causes corrupt documents. To work around this problem, the API provides an additional flag that enables you to forcibly set the input encoding, thereby overcoming an incorrect `XMLDecl`.

The precedence rules for determining input encoding are:

1. Forced encoding as specified by the user

Copyright © 1999, 2025, Oracle and/or its affiliates.

> **Note**
>
> Forced encoding can cause a fatal error if there is a conflict. For example, the input document is UTF-16 and starts with a UTF-16 BOM, but the user specifies a forced UTF-8 encoding. In this case, the parser objects about the conflict.

2. Protocol specification (HTTP header, and so on)
3. `XMLDecl` specification
4. User's default input encoding
5. The default, which is UTF-8 on ASCII platforms or UTF-E on EBCDIC platforms

After the application has determined the input encoding, it can parse the document and present the data. You are allowed to choose the presentation encoding; the data is in that encoding regardless of the original input encoding.

When an application writes back a DOM in serialized form, it can choose at that time to re-encode the presentation data. Thus, you can place the serialized document in any encoding.

5.3.2 Using NULL-Terminated and Length-Encoded C API Functions

The native string representation in C is null-terminated. Thus, the primary DOM interface takes and returns null-terminated strings. When stored in table form, however, Oracle XML DB data is *not* null-terminated but *length-encoded*. Consequently, XDK provides an additional set of length-encoded APIs for the high-frequency cases to improve performance.

In particular, the DOM functions in Table 5-5 have dual APIs.

Table 5-5 NULL-Terminated and Length-Encoded C API Functions

NULL-Terminated API	Length-Encoded API
`XmlDomGetNodeName()`	`XmlDomGetNodeNameLen()`
`XmlDomGetNodeLocal()`	`XmlDomGetNodeLocalLen()`
`XmlDomGetNodeURI()`	`XmlDomGetNodeURILen()`
`XmlDomGetNodeValue()`	`XmlDomGetNodeValueLen()`
`XmlDomGetAttrName()`	`XmlDomGetAttrNameLen()`
`XmlDomGetAttrLocal()`	`XmlDomGetAttrLocalLen()`
`XmlDomGetAttrURI()`	`XmlDomGetAttrURILen()`
`XmlDomGetAttrValue()`	`XmlDomGetAttrValueLen()`

5.3.3 Handling Errors with the C API

The C API functions typically either return a numeric error code (0 for success, nonzero on failure), or pass back an error code through a variable. In all cases, the API stores error codes. Your application can retrieve the most recent error by invoking the `XmlDomGetLastError()` function.

By default, the functions output error messages to `stderr`. However, you can register an error message callback at initialization time. When an error occurs, the application invokes the registered callback and does not print an error.

Copyright © 1999, 2025, Oracle and/or its affiliates.

5.4 Using orastream Functions

The orastream function API is an interface that enables you to stream large chunks of data out of a node instead of getting it all in one piece. Nodes of greater than 64 KB are thus accessible.

The orastream API represents a generic input or output stream. This interface is available to XDK users through `xml.h` and is defined by the `orastream` data structure and a set of functions that implement the interface. The creator of the stream passes a list of stream function addresses, along with a stream context to `OraStreamInit`. This function returns an instance of an `orastream` structure.

Several stream properties are specified at the time of initialization. If read or write is provided, the stream operates in byte mode using `OraStreamRead()` and `OraStreamWrite()`. If "read_char" or "write_char" is provided, the stream operates in character mode using `OraStreamReadChar()` and `OraStreamWriteChar()`. In character mode only complete characters are read or written and are never split over buffer boundaries.

A stream context is used to represent the state of the orastream and it persists for the lifetime of a stream.

Just like the input or output streams in Java, a source or a sink for the data is always specified. Output streams store the address of the external stream or object where they must populate the data. Similarly, input streams store the address of the object that is read.

Here are the orastream functions:

```
struct orastream;
typedef struct orastream orastream;
typedef ub4 oraerr; /* Error code: zero is success, non-zero is failure */

/* Initialize (Create) & Destroy (Terminate) stream object */

orastream  *OraStreamInit(void *sctx, void *sid, oraerr *err, ...);
oraerr     OraStreamTerm(orastream *stream);

/* Set or Change SID (streamID) for stream (returns old stream ID through osid)*/

oraerr     OraStreamSid(orastream *stream, void *sid, void **osid);

/* Is a stream readable or writable? */

boolean    OraStreamReadable(orastream *stream);
boolean    OraStreamWritable(orastream *stream);

/* Open & Close stream */

oraerr     OraStreamOpen(orastream *stream, ubig_ora *length);
oraerr     OraStreamClose(orastream *stream);

/* Read | Write byte stream */

oraerr     OraStreamRead(orastream *stream, oratext *dest, ubig_ora size,
           oratext **start, ubig_ora *nread, ub1 *eoi);
oraerr     OraStreamWrite(orastream *stream, oratext *src, ubig_ora size,
           ubig_ora *nwrote);

/* Read | Write char stream */
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
oraerr     OraStreamReadChar(orastream *stream, oratext *dest, ubig_ora size,
           oratext **start, ubig_ora *nread, ub1 *eoi);
oraerr     OraStreamWriteChar(orastream *stream, oratext *src, ubig_ora size,
           ubig_ora *nwrote);

/* Return handles for stream */

orastreamhdl *OraStreamHandle(orastream *stream);

/* Returns status: if the stream object is currently opened or not */

boolean OraStreamIsOpen(orastream *stream);
```

The stream error codes are:

```
#define ORASTREAM_ERR_NULL_POINTER      1      /* NULL pointer given */
#define ORASTREAM_ERR_BAD_STREAM        2      /* invalid stream object */
#define ORASTREAM_ERR_WRONG_DECTION     3      /* tried wrong-direction I/O */
#define ORASTREAM_ERR_UNKNOWN_PROPERTY  4      /* unknown creation prop */
#define ORASTREAM_ERR_NO_DIRECTION      5      /* neither read nor write? */
#define ORASTREAM_ERR_BI_DIRECTION      6      /* both read any write? */
#define ORASTREAM_ERR_NOT_OPEN          7      /* stream not open */
#define ORASTREAM_ERR_WRONG_MODE        8      /* wrote byte/char mode */
/* --- Open errors --- */
#define ORASTREAM_ERR_CANT_OPEN         10     /* can't open stream */
/* --- Close errors --- */
#define ORASTREAM_ERR_CANT_CLOSE        20     /* can't close stream */
```

> **See Also**
>
> *Oracle AI Database XML C API Reference* for reference information such as parameter definitions in the orastream API

Example 5-3 Using orastream Functions

```
int test_read()
{
   xmlctx *xctx = NULL;
   oratext *barray, *docName = "NSExample.xml";
   orastream* ostream = (orastream *) 0;
   xmlerr ecode = 0;
   ub4 wcount = 0;
   ubig_ora  destsize, nread;
   oraerr oerr = 0;
   ub1 eoi = 0;
   nread = destsize = 1024;
   if (!(xctx = XmlCreateNew(&ecode, (oratext *)"stream_xctx", NULL, wcount,
                             NULL)))
    {
       printf("Failed to create XML context, error %u\n", (unsigned)ecode);
       return -1;
    }

   barray = XmlAlloc(xctx, sizeof(oratext) * destsize);

   /* open function should be specified in order to read correctly. */
   if (!(ostream = OraStreamInit(NULL,docName, (oraerr *)&ecode,
                                 "open", fileopen,
                                 "read", fileread,
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
                                    NULL)))
   {
      printf("Failed to initialize OrsStream, error %u\n",(unsigned)ecode);
      return -1;
   }

   /* check readable and writable  */
    if (OraStreamReadable(ostream))
       printf("ostream is readable\n");
    else
       printf("ostream is not readable\n");

     if (OraStreamWritable(ostream))
       printf("ostream is writable\n");
    else
       printf("ostream is not writable\n");

    if (oerr = OraStreamRead(ostream, barray, destsize, &barray, &nread, &eoi))
    {
      printf("Failed to read due to orastream was not open, error %u\n", oerr);
    }

   /* open orastream */
   OraStreamOpen(ostream, NULL);

   /* read document */
   OraStreamRead(ostream, barray, destsize, &barray, &nread, &eoi);

   OraStreamTerm(ostream);

   XmlDestroy(xctx);
   return 0;
}
ORASTREAM_OPEN_F(fileopen, sctx, sid, hdl, length)
{
    FILE *fh = NULL;

    printf("Opening orastream %s...\n", (oratext *)sid);

    if (sid && ((fh= fopen(sid, "r")) != NULL))
    {
        printf("Opening orastream %s...\n", (oratext *)sid);
    }
    else
    {
         printf("Failed to open input file.\n");
         return -1;
     }

    /* store file handle generically, NULL means stdout */
    hdl->ptr_orastreamhdl = fh;

    return XMLERR_OK;
}

ORASTREAM_READ_F(fileread, sctx, sid, hdl,
                         dest, size, start, nread, eoi)
{
    FILE *fh = NULL;
    int i =0;
    printf("Reading orastream %s ...\n", (oratext *)sid);
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
    // read data from file to dest
    if ((fh = (FILE *) hdl->ptr_orastreamhdl) != NULL)
        *nread = fread(dest, 1, size, fh);
    printf("Read %d bytes from orastream...\n", (int) *nread);

    *eoi = (*nread < size);
    if (start)
        *start = dest;

    printf("printing document ...\n");
    for(i =0; i < *nread; i++)
    printf("%c", (char)dest[i]);
    printf("\nend ...\n");
    return ORAERR_OK;
}
```

5.5 Using the SAX API for C

To use SAX, initialize an `xmlsaxcb` structure with function pointers and pass it to `XmlLoadSax()`. You can also include a pointer to a user-defined context structure, which you pass to each SAX function.

> **See Also**
>
> *Oracle AI Database XML C API Reference* for the SAX callback structure

5.6 Using the XML Pull Parser for C

The XML Pull Parser is an implementation of the XML Events interface. The XML Pull Parser and the SAX parser are similar, but using the Pull Parser, the application (consumer) drives the events, while in SAX, the parser (producer) drives the events.

Both the XML Pull Parser and SAX represent the document as a sequence of events, with start tags, end tags, and comments. XML Pull Parser gives control to the application by exposing a simple set of APIs and an underlying set of events. Methods such as `XmlEvNext` allow an application to ask for (or pull) the next event, rather than handling the event in a callback, as in SAX. Thus, the application has more procedural control over XML processing. Also, the application can decide to stop further processing, unlike a SAX application, which parses the entire document.

5.6.1 Using Basic XML Pull Parsing Capabilities

The steps required to use the XML Pull Parser are described.

1. Invoke `XmlCreate` to initialize the XML meta-context.
2. Initialize the Pull Parser context by invoking the `XmlEvCreatePPCtx` function, which creates and returns the event context.

   The `XmlEvCreatePPCtx` function supports all the properties supported by `XmlLoadDom` and `XmlLoadSax`, plus some additional ones.

   The `XmlEvCreatePPCtx` and `XmlEvCreatePPCtxVA` functions are fully implemented.
3. Ensure that the event context is passed to all subsequent invocations of the Pull Parser.

Copyright © 1999, 2025, Oracle and/or its affiliates.

4. Terminate the Pull Parser context by invoking the `XmlEvDestoryPPCtx` function, to clean up memory.
5. Destroy the XML meta-context by invoking the `XmlDestoryCtx` function.

5.6.1.1 XML Event Context

The XML event context structure is shown.

Example 5-4 XML Event Context

```
typedef  struct {
   void *ctx_xmlevctx;                  /* implementation specific context */
   xmlevdisp *disp_xmlevctx;            /* dispatch table */
   ub4 checkword_xmlevctx;              /* checkword for integrity check */
   ub4 flags_xmlevctx;                  /* mode; default: expand_entity */
   struct xmlevctx *input_xmlevctx;     /* input xmlevctx; chains the XML
Event
                                           context */
} xmlevctx;
```

5.6.1.2 About the XML Event Context

Each XML Pull Parser is allowed to create its own context and implement its own API functions.

- Dispatch Table

  The dispatch table, `disp_xmlevctx`, contains one pointer for each API function, except for the `XmlEvCreatePPCtx`, `XmlEvCreatePPCtxVA`, `XmlEvDestoryPPCtx`, `XmlEvLoadPPDoc`, and `XmlEvCleanPPCtx` functions.

  When the event context is created, the pointer `disp_xmlevctx` is initialized with the address of that static table.

- Implementation-Specific Event Context

  The field `ctx_xmlevctx` must be initialized with the address of the context specific to this invocation of the particular implementation. The implementation-specific event context is of type `*void`, so that it can differ for different applications.

- Input Event Context

  Each Pull Parser can specify an input event context, `xmlevctx`. This field enables the parser to chain multiple event producers. As a result, if a dispatch function is specified as `NULL` in a context, the application uses the next non-null dispatch function in the chain of input event contexts. The base `xmlevctx` must ensure that all dispatch function pointers are non-null.

5.6.2 Parsing Multiple XML Documents

After creating and initializing the XML Event Context, an application can parse multiple documents using repeated invocations of `XmlEvLoadPPDoc` and `XmlEvCleanPPCtx`.

The properties defined by the application during the XML Event Context creation cannot be changed for each invocation of the `XmlLoadPPDoc` function. to change the properties, destroy the event context and then re-create it.

Copyright © 1999, 2025, Oracle and/or its affiliates.

After XmlEvCleanPPCtx cleans up the internal structure of the current parser, the event context can be reused to parse another document.

5.6.3 ID Callback

You can provide a callback to convert text-based names to 8-byte identifiers (IDs).

Callback Function Signature

```
typedef  sb8 (*xmlev_id_cb_funcp)( void *ctx , ub1 type, ub1 *token, ub4 tok_len,
              sb8 nmspid, boolean isAttribute);
```

Return Value

sb8: an 8-byte ID.

Arguments

- *ctx: The implementation context.
- type: The type, which is indicated by this enumeration:

  ```
  typedef enum
  {
    XML_EVENT_ID_URI,
    XML_EVENT_ID_QNAME,
  }xmlevidtype;
  ```

- *token and tok_len: The actual text to be converted.
- nmspid: The namespace ID.
- isAttribute: A Boolean value indicating an attribute.

Internally, the XmlEvGetTagId and XmlEvGetAttrID APIs invoke this callback twice, once to fetch the namespace ID and once to fetch the actual ID of the tag or the attribute Qname.

The XmlEvGetTagUriID and XmlEvGetAttrUriID functions invoke this callback once to get the ID of the corresponding Universal Resource Identifier (URI).

If a callback is not supplied, an error XML_ERR_EVENT_NOIDCBK is returned when these APIs are used.

5.6.4 Error Handling for the XML Pull Parser

Error handling for the XML Pull Parser is described.

5.6.4.1 Parser Errors

Errors raised by the parser are described.

The XML Pull Parser returns the message XML_EVENT_FATAL_ERROR when it throws an error because the input document is malformed. Function XmlEvGetError is provided to get the error number and message.

During the XmlEvCreatePPCtx operation, any error handler supplied by the application during XmlCreate is overridden. The application must invoke the XmlErrSetHandler function after the XmlEvDestroyPPCtx operation to restore the original callback.

Copyright © 1999, 2025, Oracle and/or its affiliates.

5.6.4.2 Programming Errors

To handle programmatic errors. XDK provides a callback that the application can supply when creating an event context. This callback is invoked when the application invokes an illegal API.

The callback signature is:

```
typedef  void (* xmlev_err_cb_funcp)(xmlctx *xctx, xmlevctx *evctx,
        xmlevtype cur_event);
```

An example of an illegal API invocation is:

`XmlEvGetName` cannot be called for the `XML_EVENT_CHARACTERS` event.

5.6.5 Sample Pull Parser Application

A sample pull parser application, a document to be parsed, and a list of the events that the application generates from the document are presented.

Example 5-5 shows the sample application code.

Example 5-6 shows the sample document to be parsed.

Example 5-7 shows the sequence of events generated when the attribute events property is `FALSE` and the expand entities properties is `TRUE`.

Example 5-5 Sample Pull Parser Application Example

```
# include "xml.h"
# include "xmlev.h"
...
xmlctx *xctx;
xmlevctx *evtcx;
if (!(xctx = XmlCreate(&xerr, (oratext *) "test")))
{
    printf("Failed to create XML context, error %u\n", (unsigned) xerr);
    return -1;
}
...
if(!(evctx = XmlEvCreatePPCtx(xctx, &xerr, NULL)))
{
   printf("Failed to create EVENT context, error %u\n", (unsigned) xerr);
   return -1;
 }
for(i = 0; i < numDocs; i++)
{
  if (xerr = XmlEvLoadPPDoc(xctx, evctx, "file", input_filenames[i], 0, NULL)
     {
       printf("Failed to load the document, error %u\n", (unsigned) xerr);
       return -1;
     }
...
  for(;;)
  {
    xmlevtype cur_event;
    cur_event = XmlEvNext(evctx);
    switch(cur_event)
         {
               case XML_EVENT_FATAL_ERROR:
                     XmlEvGetError(evctx, (oratext **)&errmsg);
                          printf("Error %s\n", errmsg);
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
                return;
                case XML_EVENT_START_ELEMENT:
                      printf("<%s>", XmlEvGetName0(evctx));
                break;
                case XML_EVENT_END_DOCUMENT:
                      printf("<%s>", XmlEvGetName0(evctx));
                return;
          }
  }
  XmlEvCleanPPCtx(xctx, evctx);
}
XmlEvDestroyPPCtx(xctx, evctx);
XmlDestroy(xctx);
```

Example 5-6 Sample Document to Parse

```
<!DOCTYPE doc [
<!ENTITY ent SYSTEM "file:attendees.txt">
<!ELEMENT doc ANY>
<!ELEMENT meeting (topic, date, publishAttendees)>
<!ELEMENT publishAttendees (#PCDATA)>
<!ELEMENT topic (#PCDATA)>
<!ELEMENT date (#PCDATA)>
]>
<!-- Begin Document -->
<doc>
  <!-- Info about the meeting -->
  <meeting>
    <topic>Group meeting</topic>
    <date>April 25, 2005</date>
    <publishAttendees>&ent;</publishAttendees>
  </meeting>
</doc>
<!-- End Document -->
```

Example 5-7 Events Generated by Parsing a Sample Document

```
XML_EVENT_START_DOCUMENT
XML_EVENT_START_DTD
XML_EVENT_PE_DECLARATION
XML_EVENT_ELEMENT_DECLARATION
XML_EVENT_ELEMENT_DECLARATION
XML_EVENT_ELEMENT_DECLARATION
XML_EVENT_ELEMENT_DECLARATION
XML_EVENT_ELEMENT_DECLARATION
XML_EVENT_END_DTD
XML_EVENT_COMMENT
XML_EVENT_START_ELEMENT
XML_EVENT_SPACE
XML_EVENT_COMMENT
XML_EVENT_SPACE
XML_EVENT_START_ELEMENT
XML_EVENT_START_ELEMENT
XML_EVENT_CHARACTERS
XML_EVENT_END_ELEMENT
XML_EVENT_START_ELEMENT
XML_EVENT_CHARACTERS
XML_EVENT_END_ELEMENT
XML_EVENT_START_ELEMENT
XML_EVENT_START_ENTITY
XML_EVENT_CHARACTERS
XML_EVENT_END_ENTITY
XML_EVENT_END_ELEMENT
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
XML_EVENT_END_ELEMENT
XML_EVENT_SPACE
XML_EVENT_END_ELEMENT
XML_EVENT_COMMENT
XML_EVENT_END_DOCUMENT
```

5.7 Using OCI and the XDK for C API

This section describes accessing XDK for C functions from Oracle Call Interface (OCI).

5.7.1 Using XMLType Functions and Descriptions

You can use the C API for XML with `XMLType` columns in the database. An Oracle Call Interface (OCI) program can access XML data stored in a table by initializing the values of OCI handles.

This applies to handles such as these:

- Environment handle
- Service handle
- Error handle
- Optional parameters

The program can pass these input values to the function `OCIXmlDbInitXmlCtx()`, which returns an XML context. After the program invokes the C API, the function `OCIXmlDbFreeXmlCtx()` frees the context.

Table 5-6 XMLType Functions

Function Name	Description
`XmlCreateDocument()`	Create empty `XMLType` instance
`XmlLoadDom()` and so on	Create from a source buffer
`XmlXPathEvalexpr()` and family	Extract an `XPath` expression
`XmlXslProcess()` and family	Transform using an Extensible Stylesheet Language Transformation (XSLT) stylesheet
`XmlXPathEvalexpr()` and family	Check if an `XPath` exists
`XmlDomIsSchemaBased()`	Is document schema-based?
`XmlDomGetSchema()`	Get schema information
`XmlDomGetNodeURI()`	Get document namespace
`XmlSchemaValidate()`	Validate using schema
Cast `(void *)` to `(xmldocnode *)`	Get DOM from `XMLType`
Cast `(xmldocnode *)` to `(void *)`	Get `XMLType` from DOM

5.7.2 Initializing an XML Context for Oracle XML DB

An XML context is a required parameter in each C DOM API function. This opaque context encapsulates information pertaining to data encoding, error message language, and so on.

Copyright © 1999, 2025, Oracle and/or its affiliates.

The contents of this XML context are different for XDK applications and for Oracle XML DB applications.

> **Note**
>
> Do not use an XML context for XDK in an Oracle XML DB application, or an XML context for Oracle XML DB in an XDK application.

For Oracle XML DB, the two OCI functions that initialize and free an XML context have these prototypes:

```
xmlctx *OCIXmlDbInitXmlCtx (OCIEnv *envhp, OCISvcCtx *svchp, OCIError *errhp,
       ocixmldbparam *params, ub4 num_params);

void OCIXmlDbFreeXmlCtx (xmlctx *xctx);
```

> **See Also**
>
> - *Oracle Call Interface Developer's Guide* for reference material on the functions
> - *Oracle Call Interface Developer's Guide* for a discussion about OCI support for XML
> - *Oracle AI Database XML C API Reference* for reference information on the DOM APIs

5.7.3 Creating XMLType Instances on the Client

You can construct new `XMLType` instances on the client by using the `XmlLoadDom()` invocations.

Follow these basic steps:

1. You must initialize the `xmlctx`, as showd in the example in Using the DOM API for C.
2. You can construct the XML data itself from these sources:
   - User buffer
   - Local file
   - URI

   The return value from these is an `(xmldocnode *)`, which you can use in the rest of the common C API.
3. You can cast the `(xmldocnode *)` to a `(void *)` and directly provide it as the bind value if required.

You can construct empty `XMLType` instances by invoking `XmlCreateDocument()`. This function would be equivalent to an `OCIObjectNew()` for other types. You can operate on the `(xmldocnode *)` returned by the preceding invocation and finally cast it to a `(void *)` if it must be provided as a bind value.

Copyright © 1999, 2025, Oracle and/or its affiliates.

5.7.4 Operating on XML Data in the Database Server

You can operate on XML data in Oracle Database using OCI statements. You can bind and define `XMLType` values using `xmldocnode` and use OCI statements to extract XML data from the database. You can use this data directly in C DOM functions or bind values directly to SQL statements.

5.7.5 Using OCI and the XDK for C API: Examples

Examples show how to use the DOM API to construct and save an XML schema-based document and to modify a database document.

Example 5-8 shows how to construct a schema-based document with the DOM API and save it to the database. You must include the header files `xml.h` and `ocixmldb.h`.

Example 5-9 shows how to get a document from the database and modify it with the DOM API.

Example 5-8 Constructing a Schema-Based Document with the DOM API

```
#include <xml.h>
#include <ocixmldb.h>
static oratext tlpxml_test_sch[] = "<TOP xmlns='example1.xsd'\n\
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance' \n\
xsi:schemaLocation='example1.xsd example1.xsd'/>";

void example1()
{
    OCIEnv *envhp;
    OCIError *errhp;
    OCISvcCtx *svchp;
    OCIStmt *stmthp;
    OCIDuration dur;
    OCIType *xmltdo;

    xmldocnode  *doc;
    ocixmldbparam params[1];
    xmlnode *quux, *foo, *foo_data;
    xmlerr       err;

    /* Initialize envhp, svchp, errhp, dur, stmthp */
    /* ........ */

    /* Get an xml context */
    params[0].name_ocixmldbparam = XCTXINIT_OCIDUR;
    params[0].value_ocixmldbparam = &dur;
    xctx = OCIXmlDbInitXmlCtx(envhp, svchp, errhp, params, 1);

    /* Start processing */
    printf("Supports XML 1.0: %s\n",
       XmlHasFeature(xctx, (oratext *) "xml", (oratext *) "1.0") ? "YES" :
"NO");

    /* Parsing a schema-based document */
    if (!(doc = XmlLoadDom(xctx, &err, "buffer", tlpxml_test_sch,
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
                        "buffer_length", sizeof(tlpxml_test_sch)-1,
                        "validate", TRUE, NULL)))
    {
       printf("Parse failed, code %d\n");
       return;
    }

    /* Create some elements and add them to the document */
    top = XmlDomGetDocElem(xctx, doc);
    quux = (xmlnode *) XmlDomCreateElem(xctx ,doc, (oratext *) "QUUX");
    foo = (xmlnode *) XmlDomCreateElem(xctx, doc, (oratext *) "FOO");
    foo_data = (xmlnode *) XmlDomCreateText(xctx, doc, (oratext *)"foo's
data");
    foo_data = XmlDomAppendChild(xctx, (xmlnode *) foo, (xmlnode *) foo_data);
    foo = XmlDomAppendChild(xctx, quux, foo);
    quux = XmlDomAppendChild(xctx, top, quux);

    XmlSaveDom(xctx, &err, top, "stdio", stdout, NULL);
    XmlSaveDom(xctx, &err, doc, "stdio", stdout, NULL);

    /* Insert the document to my_table */
    ins_stmt = "insert into my_table values (:1)";

    status = OCITypeByName(envhp, errhp, svchp, (const text *) "SYS",
                  (ub4) strlen((char *)"SYS"), (const text *) "XMLTYPE",
                  (ub4) strlen((char *)"XMLTYPE"), (CONST text *) 0,
                  (ub4) 0, dur, OCI_TYPEGET_HEADER,
                  (OCIType **) &xmltdo)) ;

    if (status == OCI_SUCCESS)
    {
       exec_bind_xml(svchp, errhp, stmthp, (void *)doc, xmltdo, ins_stmt));
    }

   /* free xml ctx */
   OCIXmlDbFreeXmlCtx(xctx);
}

/*-------------------------------------------------------*/
/* execute a sql statement which binds xml data */
/*-------------------------------------------------------*/
sword exec_bind_xml(svchp, errhp, stmthp, xml, xmltdo, sqlstmt)
OCISvcCtx *svchp;
OCIError *errhp;
OCIStmt *stmthp;
void *xml;
OCIType *xmltdo;
OraText *sqlstmt;
{
   OCIBind *bndhp1 = (OCIBind *) 0;
   OCIBind *bndhp2 = (OCIBind *) 0;
   sword  status = 0;
   OCIInd ind = OCI_IND_NOTNULL;
   OCIInd *indp = &ind;

   if(status = OCIStmtPrepare(stmthp, errhp, (OraText *)sqlstmt,
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
                    (ub4)strlen((char *)sqlstmt),
                    (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT)) {
      return OCI_ERROR;
    }

    if(status = OCIBindByPos(stmthp, &bndhp1, errhp, (ub4) 1, (dvoid *) 0,
                    (sb4) 0, SQLT_NTY, (dvoid *) 0, (ub2 *)0,
                    (ub2 *)0, (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT)) {
      return OCI_ERROR;
    }

    if(status = OCIBindObject(bndhp1, errhp, (CONST OCIType *) xmltdo,
                (dvoid **) &xml, (ub4 *) 0, (dvoid **) &indp, (ub4 *) 0)) {
      return OCI_ERROR;
    }

    if(status = OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                 (CONST OCISnapshot*) 0, (OCISnapshot*) 0, (ub4) OCI_DEFAULT))
{
      return OCI_ERROR;
  }

    return OCI_SUCCESS;
}
```

Example 5-9 Modifying a Database Document with the DOM API

```
#include <xml.h>
#include <ocixmldb.h>
sword example2()
{
    OCIEnv *envhp;
    OCIError *errhp;
    OCISvcCtx *svchp;
    OCIStmt *stmthp;
    OCIDuration dur;
    OCIType *xmltdo;

    xmldocnode  *doc;
    xmlnodelist *item_list; ub4 ilist_l;
    ocixmldbparam params[1];
    text *sel_xml_stmt = (text *)"SELECT xml_col FROM my_table";
    ub4    xmlsize = 0;
    sword  status = 0;
    OCIDefine *defnp = (OCIDefine *) 0;

    /* Initialize envhp, svchp, errhp, dur, stmthp */
    /* ... */

    /* Get an xml context */
    params[0].name_ocixmldbparam = XCTXINIT_OCIDUR;
    params[0].value_ocixmldbparam = &dur;
    xctx = OCIXmlDbInitXmlCtx(envhp, svchp, errhp, params, 1);

    /* Start processing */
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
    if(status = OCITypeByName(envhp, errhp, svchp, (const text *) "SYS",
                  (ub4) strlen((char *)"SYS"), (const text *) "XMLTYPE",
                  (ub4) strlen((char *)"XMLTYPE"), (CONST text *) 0,
                  (ub4) 0, dur, OCI_TYPEGET_HEADER,
                  (OCIType **) xmltdo_p)) {
       return OCI_ERROR;
    }

    if(!(*xmltdo_p)) {
       printf("NULL tdo returned\n");
       return OCI_ERROR;
    }

    if(status = OCIStmtPrepare(stmthp, errhp, (OraText *)selstmt,
                   (ub4)strlen((char *)selstmt),
                   (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT)) {
      return OCI_ERROR;
    }

    if(status = OCIDefineByPos(stmthp, &defnp, errhp, (ub4) 1, (dvoid *) 0,
                  (sb4) 0, SQLT_NTY, (dvoid *) 0, (ub2 *)0,
                  (ub2 *)0, (ub4) OCI_DEFAULT)) {
       return OCI_ERROR;
    }

    if(status = OCIDefineObject(defnp, errhp, (OCIType *) *xmltdo_p,
                           (dvoid **) &doc,
                           &xmlsize, (dvoid **) 0, (ub4 *) 0)) {
      return OCI_ERROR;
    }

    if(status = OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                 (CONST OCISnapshot*) 0, (OCISnapshot*) 0, (ub4)
OCI_DEFAULT)) {
      return OCI_ERROR;
    }

    /* We have the doc. Now we can operate on it */
    printf("Getting Item list...\n");

   item_list = XmlDomGetElemsByTag(xctx,(xmlelemnode *) elem,(oratext
*)"Item");
    ilist_l   = XmlDomGetNodeListLength(xctx, item_list);
    printf(" Item list length = %d \n", ilist_l);

    for (i = 0; i < ilist_l; i++)
    {
      elem = XmlDomGetNodeListItem(xctx, item_list, i);
      printf("Elem Name:%s\n", XmlDomGetNodeName(xctx, fragelem));
      XmlDomRemoveChild(xctx, fragelem);
    }

    XmlSaveDom(xctx, &err, doc, "stdio", stdout, NULL);

   /* free xml ctx */
   OCIXmlDbFreeXmlCtx(xctx);
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
    return OCI_SUCCESS;
}
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

6

Using Binary XML with C

An explanation is given of how to use binary Extensible Markup Language (binary XML) with C.

6.1 Introduction to Binary XML for C

Client-side processing of Extensible Markup Language (XML) data can use either `XMLType` data stored in the database, including data in binary XML format, or transient data that is not in the database.

6.2 Prerequisites for Using Binary XML with C

This chapter assumes that you are familiar with the XML Parser for C, the basic concepts of binary XML, and the OCI (Oracle Call Interface). Only the OCI API can be used for programming in C with binary XML.

Related Topics

- Using the XML Parser for C
  An explanation is given of how to use the Extensible Markup Language (XML) parser for C.
- Using Binary XML with Java
  Topics here explain how to use Binary XML with Java.

See Also

- *Oracle XML DB Developer's Guide*
- *Oracle Call Interface Developer's Guide*

6.3 Binary XML Storage Format – C

Binary XML is an optimized format for XML. It includes encoding and decoding of XML documents, from text to binary and binary to text. Binary XML is XML Schema-aware, but it can also be used for XML data that is not based on an XML schema.

A binary XML processor is a component that processes and transforms binary XML format into text and XML text into binary XML format.

The mid-tier and client tiers can produce, consume, and process XML in binary XML format. The C application fetches data from Oracle XML DB Repository, performs updates on the XML using DOM, and stores it back in the database. Or an XML document is created or input on the client and XSLT, XQuery, and other utilities can be used on it. Then the output XML is saved in Oracle XML DB. Further details of concepts and reference pages for OCI functions are described in the Oracle Call Interface Programmer's Guide.

Copyright © 1999, 2025, Oracle and/or its affiliates.

7

Using the XML Schema Processor for C

An explanation is given of how to use the Extensible Markup Language (XML) schema processor for C.

> **Note**
>
> Use the unified C application programming interface (API) for Oracle XML Developer's Kit (XDK) and Oracle XML DB applications. Older, nonunified C functions are deprecated and supported only for backward compatibility. They will be removed in a future release.
>
> The unified C API is described in Overview of the Unified C API.

7.1 Oracle XML Schema Processor for C

The XML Schema processor for C is a companion component to the Extensible Markup Language (XML) parser for C that allows support for simple and complex data types in XML applications.

The XML Schema processor for C supports the World Wide Web Consortium (W3C) XML Schema Recommendation. This makes writing custom applications that process XML documents straightforward, and means that a standards-compliant XML Schema processor is part of XDK on every operating system where Oracle Database is ported.

The XML Schema processor enables validation of XML and retrieval of metadata. It can be called by itself or through the XML Parser for C.

> **See Also**
>
> XML Parsing for Java, for more information about XML Schema and why you would want to use XML Schema.

7.1.1 Oracle XML Schema for C Features

The features of the Oracle XML Schema processor for C are described.

Features:

- Supports simple and complex types
- Built on XML parser for C
- Supports the W3C XML Schema Recommendation

Copyright © 1999, 2025, Oracle and/or its affiliates.

> **See Also**
>
> - *Oracle AI Database XML C API Reference* "Schema APIs for C"
> - `$ORACLE_HOME/xdk/demo/c/schema/` - sample code

7.1.2 Standards Conformance for Oracle XML Schema Processor for C

The standards to which the XML Schema Processor for C conforms are listed.

- W3C recommendation for Extensible Markup Language (XML) 1.0
- W3C recommendation for Document Object Model (DOM) Level 1.0
- W3C recommendation for Namespaces in XML
- W3C recommendation for XML Schema

7.1.3 XML Schema Processor for C: Supplied Software

The software supplied for the XML Schema Processor for C is described.

Table 7-1 XML Schema Processor for C: Supplied Files in $ORACLE_HOME

Directory and Files	Description
`bin`	schema processor executable, `schema`
`lib`	XML/XSL/Schema & support libraries
`nls/data`	Globalization Support data files
`xdk/demo/c/schema`	example usage of the Schema processor
`xdk/include`	header files
`xdk/mesg`	error message files
`xdk/readme.html`	introductory file

Table 7-2 lists the included libraries in directory `lib`.

Table 7-2 XML Schema Processor for C: Supplied Libraries

Included Library	Description
`libxml21.a`	XML parser, Extensible Stylesheet Language Transformation (XSLT) processor, XML Schema processor
`libcore21.a`	Common Oracle Runtime Environment (CORE) functions
`libnls21.a`	Globalization Support

7.2 Using the C XML Schema Processor Command-Line Utility

You can call XML Schema processor for C as an executable by invoking `bin/schema` in the install area.

The executable takes two arguments:

Copyright © 1999, 2025, Oracle and/or its affiliates.

- XML instance document
- Optionally, a default schema

XML Schema processor for C 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 `xdk/demo/c/schema` subdirectory for details on how to build your program.

Error message files in different languages are provided in the `mesg/` subdirectory.

7.3 XML Schema Processor for C Usage Diagram

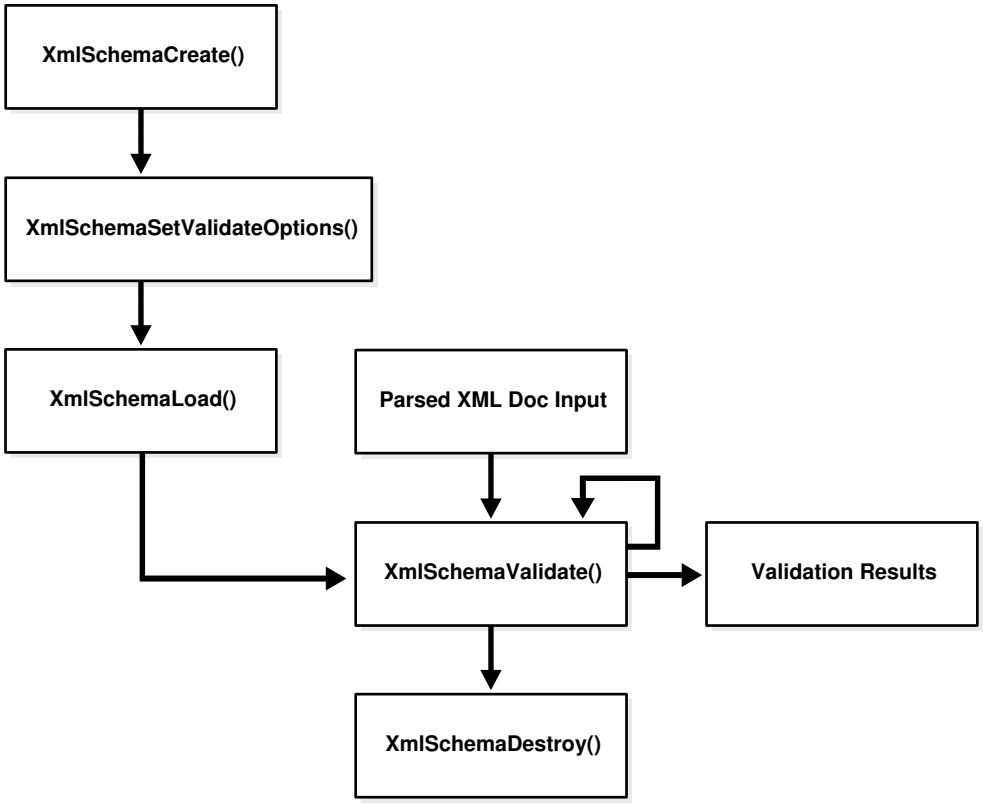
The calling sequence for the XML Schema processor for C is presented.

Figure 7-1illustrates the calling sequence, which is as follows:

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. Schema documents to be used in the session are loaded in advance.
3. The instance document to be validated is first parsed with the XML parser.
4. The top of the XML element subtree for the instance is then passed to the schema validate function.
5. If no explicit schema is defined in the instance document, any loaded schemas are used.
6. More documents can then be validated using the same schema context.
7. When the session is over, the Schema tear-down function is called, which releases all memory allocated for the loaded schemas.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Figure 7-1 XML Schema Processor for C Usage Diagram



7.4 How to Run XML Schema for C Sample Programs

Directory `xdk/demo/c/schema` contains sample XML Schema applications that show how to use Oracle XML Schema processor with its API. These sample files are described here.

Table 7-3 XML Schema for C Samples Provided

Sample File	Description
`Makefile`	Makefile to build the sample programs and run them, verifying correct output.
`xsdtest.c`	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.
`aq.{xsd,xml,std}`	Second sample schema, instance document, and expected output respectively, after running `xsdtest` on them.
`pub.{xsd,xml,std}`	Third sample schema, instance document, and expected output respectively, after running `xsdtest` on them.

To build the sample programs, run `make`.

To build the programs and run them, comparing the actual output to expected output:

```
make sure
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

7.5 What Is the Streaming Validator?

The streaming validator uses XML Events, which is a representation of an XML document that is similar to Simple API for XML (SAX) Events. XML events has a start tag, end tag, and comment. The producer drives the SAX events and the consumer drives the XML events.

The streaming validator shares software with the older schema validator and derives most functionality from it. Memory overhead is less than for the DOM representation used in the older validator. Only one pass is made over the document. The streaming validator was introduced in Oracle Database 11g Release 1 (11.1).

There are two modes of streaming validation:

- Transparent mode—events are returned to the application.
- Opaque mode—events are not returned to the application but an error indicating success or failure of the document validation process is returned.

Before document validation, the regular validation context must be created, and the relevant schema must be loaded using this context. Then XML event context for pull parser (or for another event producer) must be created. This event context is then given to the streaming validator, so that it can request events from the producer.

Passing in a schema DOM to the `XmlSchemaLoad` API is also supported.

7.5.1 Using Transparent Mode

Basic use of transparent mode is described.

An application starts by invoking `XmlEvCreateSVCtx()`. This invocation creates and returns an event context of type `xmlctx`, which must be passed on all subsequent invoking the streaming validator. The event context created must be terminated by invoking `XmlEvDestroyCtx()`.

After creation of the event context, the application repeatedly advances validation to the next event by invoking `XmlEvNext()`, which returns the type of the next event. Additional API interfaces allow the application to retrieve information relevant to the last event.

7.5.1.1 Error Handling in Transparent Mode

There is no notion of a valid event. Validity is the property of a document and not of the individual items and events of the document.

The errors are:

- `XML_EVENT_FATAL_ERROR`—When the producer of XML events reports this error, the streaming validator returns this event back to the application and stops the validation process.
- `XML_EVENT_ERROR`—The streaming validator returns this event to the application when a validation error occurs. The application can then invoke `XmlEvGetError()` to get more information about the error.

If the application does not receive any `XML_EVENT_ERROR` or `XML_EVENT_FATAL_ERROR` events, the document is valid. Therefore, the application must handle these events and not ignore them.

These errors are not cached and the associated information is not available for later retrieval.

Copyright © 1999, 2025, Oracle and/or its affiliates.

7.5.1.2 Streaming Validator Example

A streaming validator example in transparent mode is presented.

Example 7-1 Streaming Validator in Transparent Mode

```
# include "xmlev.h"
...
xmlevctx *ppevtcx, *svevctx;
xmlctx *xctx
xsdctx *sctx;

if (!(xctx = XmlCreate(&xerr, (oratext *) "test")))
    printf("Failed to create XML context, error %u\n",
                        (unsigned) xerr);
...
if (!(sctx = XmlSchemaCreate(xctx, &xerr, NULL)))
    printf("Failed to create schema context, error %u\n",
                        (unsigned) xerr);

...
If (xerr = XmlSchemaLoad(sctx, "my_schema.xsd", NULL))
    printf("Failed to load schema, error %u\n",
                        (unsigned) xerr);

if(!(ppevctx = XmlEvCreatePPCtx(xctx, &xerr, NULL)))
   printf("Failed to create EVENT context, error %u\n",
                        (unsigned) xerr);

if(xerr = XmlEvLoadPPDoc(xctx, ppevctx, "file", "test.xml", 0, NULL))
   printf("Failed to load Document, error %u\n",
                        (unsigned) xerr);

...
If(!(svevctx = XmlEvCreateSVCtx(xctx, sctx, ppevctx, &xerr)))
   printf("Failed to create SVcontext, error %u\n",
                        (unsigned) xerr);
...
for(;;)
{
   xmlevtype cur_event;
   cur_event = XmlEvNext(svevctx);
   switch(cur_event)
    {
        case XML_EVENT_FATAL_ERROR:
           printf("FATAL ERROR");
           /* error processing goes here */
           return;
        case XML_EVENT_ERROR:
           XmlEvGetError(svevctx, oratext *msg);
           printf("Validation Failed, Error %s\n", msg);
           break;
        case XML_EVENT_START_ELEMENT:
           printf("<%s>", XmlEvGetName(svevctx));
           break;
...
        case XML_EVENT_END_DOCUMENT:
           printf("END DOCUMENT");
           return;
    }
}
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
...
XmlEvDestroySVCtx(svevctx);
XmlSchemaDestroy(sctx);
XmlEvDestroyCtx(ppevctx);
XmlDestroyCtx(xctx);
```

7.5.2 Using Opaque Mode

In opaque mode, the streaming validator reads the instance document to be validated as a sequence of events from the producer, but it does not pass the events to the application (consumer). It returns `XMLERR_OK` on success and an error number on failure.

After the schema has been loaded and the XML events context has been initialized, an application can validate the document in this mode by invoking `XmlEvSchemaValidate()`. The signature of this function takes a pointer to the events context. The declaration is:

```
xmlerr XmlEvSchemaValidate(xmlctx *xctx, xsdctx *sctx, xmlevctx *evctx,
       oratext **errmsg);
/* Returns (xmlerr), the error code */
```

7.5.2.1 Error Handling in Opaque Mode

When the streaming validator encounters an error, `XmlEvSchemaValidate()` returns an error number. This could be because of a parse error or a validation error. The application can then use the existing `XmlEvGetError` APIs to get the error message.

The error message is parameterized and typically has all of the errors leading up to the point where the streaming validator terminated.

7.5.2.2 Example of Opaque Mode Application

An example of opaque mode application is presented.

Example 7-2 Example of Streaming Validator in Opaque Mode

```
# include "xmlev.h"
...
xmlevctx *ppevtcx;
xmlctx   *xctx;
xsdctx   *sctx;
oratext  **errmsg;
xmlerr   xerr;

if (!(xctx = XmlCreate(&xerr, (oratext *) "test"))
      printf("Failed to create XML context, error %u\n", (unsigned) xerr);
...
if (!(sctx = XmlSchemaCreate(xctx, &xerr, NULL)))
      printf("Failed to create schema context, error %u\n", (unsigned) xerr);

...
if (xerr = XmlSchemaLoad(sctx, "my_schema.xsd", NULL))
    printf("Failed to load schema, error %u\n", (unsigned) xerr);

if(!(ppevctx = XmlEvCreatePPCtx(xctx, &xerr, NULL)))
     printf("Failed to create EVENT context, error %u\n", (unsigned) xerr);

if(xerr = XmlEvLoadPPDoc(xctx, ppevctx, "file", "test.xml", 0, NULL))
   printf("Failed to load Document, error %u\n", (unsigned) xerr);
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
if((xerr = XmlEvSchemaValidate(xctx, sctx, ppevctx, errmsg)))
{
  printf("Validation Failed, Error: %s\n", errmsg);
}
...
XmlSchemaDestroy(sctx);
XmlEvDestroyCtx(ppevctx);
XmlDestroyCtx(xctx);
```

7.5.3 Using Function XmlSchemaLoad() With an Existing DOM

Function `XmlSchemaLoad()` accepts two fixed arguments and a set of variable properties. The first argument is the schema context; the second is the URL location of the schema document.

Starting with Oracle Database 11g Release 1 (11.1), you can use property `schema_dom_callback` to provide access to the schema DOM given a URL. The property is a callback function provided by the application. If supplied, the schema load function uses this callback to access the DOM for the main schema and to access any included, imported, or redefined schemas.

The callback signature is as follows:

```
typedef  xmldocnode* (*xmlsch_dom_callback) (xmlctx *xctx, oratext *uri,
         xmlerr *xerr);
```

The callback accepts a URI (the schema load function passes in the URI of the document desired) and returns the document node. Example 7-3 illustrates this.

Example 7-3 XmlSchemaLoad() Example

```
# include "xmlev.h"
...
xmlctx *xctx;
xsdctx *sctx;
xmldocnode *doc;

if (!(xctx = XmlCreate(&xerr, (oratext *) "test"))
    printf("Failed to create XML context, error %u\n", (unsigned) xerr);
...
if (!(sctx = XmlSchemaCreate(xctx, &xerr, NULL)))
    printf("Failed to create schema context, error %u\n", (unsigned) xerr);
...
If (xerr = XmlSchemaLoad(sctx, schema_uri, "schema_dom_callback", func1,  NULL))
    printf("Failed to load schema, error %u\n", (unsigned) xerr);
...
XmlSchemaDestroy(sctx);
XmlDestroyCtx(xctx);
```

7.5.4 Validation Options

You can supply options to the validation process using `XmlSchemaSetValidateOptions()`.

For example:

```
XmlSchemaSetValidateOptions(scctx, "ignore_id_constraint", (boolean)TRUE,
NULL);
```

The options are:

- `ignore_id_constraint` (existing before Oracle Database 11g Release 1 (11.1))

Copyright © 1999, 2025, Oracle and/or its affiliates.

- `ignore_sch_location` (existing before Oracle Database 11g Release 1 (11.1))
- `ignore_par_val_rest` (existing before Oracle Database 11g Release 1 (11.1))
- `ignore_pattern_check`: When this property is `TRUE`, the streaming validator ignores pattern-facet checks. The default is `FALSE`.
- `no_events_for_defaults`: When this property is `TRUE`, the streaming validator does not return events for default values added to the instance document. *This option can be used only in the transparent case.*

Example 7-4 Example of Streaming Validator Using New Options

```
# include "xmlev.h"
...
xmlevctx *ppevtcx;
xmlctx   *xctx;
xsdctx   *sctx;
xmlerr   xerr;
oratext  **errmsg;

if (!(xctx = XmlCreate(&xerr, (oratext *) "test"))
    printf("Failed to create XML context, error %u\n", (unsigned) xerr);
...
if (!(sctx = XmlSchemaCreate(xctx, &xerr, NULL)))
    printf("Failed to create schema context, error %u\n", (unsigned) xerr);
...
If (xerr = XmlSchemaLoad(sctx, "my_schema.xsd", NULL))
    printf("Failed to load schema, error %u\n", (unsigned) xerr);
if(!(ppevctx = XmlEvCreatePPCtx(xctx, &xerr, "file", "test.xml", NULL)))
   printf("Failed to create EVENT context, error %u\n", (unsigned) xerr);

if(xerr = XmlEvLoadPPDoc(xctx, ppevctx, "file", "test.xml", 0, NULL))
   printf("Failed to load Document, error %u\n", (unsigned) xerr);

XmlSchemaSetValidateOptions(sctx, "ignore_id_constraint", TRUE,
                                  "ignore_pattern_facet", TRUE, NULL);
if((xerr = XmlEvSchemaValidate(xctx,sctx, ppevctx, errmsg)))
{
  printf("Validation Failed, Error: %s\n", errmsg);
}
...
XmlSchemaDestroy(sctx);
XmlEvDestroyCtx(ppevctx);
XmlDestroyCtx(xctx);
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

8

Determining XML Differences Using C

An explanation is given of how to determine the differences between two Extensible Markup Language (XML) inputs and apply the differences as a patch to one of the XML documents.

8.1 Overview of XMLDiff in C

You can use Oracle `XmlDiff` to determine the differences between two similar XML documents. It generates an `Xdiff` instance document that indicates the differences. The `Xdiff` document is an XML document that conforms to an XML schema, the `Xdiff` schema.

You can then use `XmlPatch`, which takes the `Xdiff` instance document and applies the changes to other documents. You can use this process to apply the same changes to a large number of XML documents.

`XmlDiff` supports only the Document Object Model (DOM) application programming interface (API) for input and output.

`XmlPatch` also supports the DOM for the input and patch documents.

You can use `XmlDiff` and `XmlPatch` through a C API or a command-line tool. They are exposed by two structured query language (SQL) functions.

An `XmlHash` C API is provided to compute the hash value of an XML tree or subtree. If hash values of two trees or subtrees are equal, the trees are identical to a very high probability.

8.1.1 Process Flow for XMLDiff

The XMLDiff process flow is described.

1. The two input documents are compared by `XmlDiff`.
2. `XmlDiff` creates a `Xdiff` instance document.
3. The application can pass the `Xdiff` instance document to `XmlPatch`, if this is required.
4. `XmlPatch` can apply the differences captured from the comparison to other documents as specified by the application.

8.2 Using XmlDiff

`XmlDiff` compares the trees that represent two input documents, to determine their differences. Both input documents must use the same character-set encoding. The `Xdiff` (output) instance document has the same encoding as the data encoding (DOM encoding) of the input documents.

8.2.1 User Options for Comparison Optimization

There are two optimization options for comparison: global and local optimization.

- Global Optimization—Default

Copyright © 1999, 2025, Oracle and/or its affiliates.

The whole document trees are compared.

- Local Optimization

  Comparison is at the sibling level. Local optimization compares siblings under the corresponding parents from two trees.

Global optimization can take more time and space for large documents but always produces the smallest set of differences (the optimal difference). Local optimization is much faster, but may not produce the optimal difference.

8.2.2 User Option for Hashing

Hashing generally speeds up global optimization with a small possible loss in quality. Hashing improves the quality of the difference output, with local optimization. Using different hash levels may generate both local and global differences. You can specify the use of hashing for both local and global optimization.

To specify hashing, provide the `hashLevel` parameter. If `hashLevel` is greater than 1, then only the `DOMHash` values are used for comparing all subtrees at `depth >= hashLevel` of difference. If the hash values are equal, then the subtrees are presumed to be equal.

8.2.3 How XmlDiff Looks at Input Documents

How `XmlDiff` handles input documents is described.

`XmlDiff` ignores differences in the order of attributes while doing the comparison.

`XmlDiff` ignores `DocType` declarations. Files are not validated against the document type definition (DTD).

`XmlDiff` ignores any differences in the namespace prefixes if the namespace prefixes refer to the same namespace Universal Resource Identifier (URI). Otherwise, if two nodes have the same local name and content but differ in namespace URI, these differences are indicated.

> **Note**
>
> `XmlDiff` operates on its input documents in a nonschema-based way. It does not operate on elements or attributes in a type-aware manner.

8.2.4 Using the XmlDiff Command-Line Utility

The command-line options for utility `XmlDiff` are described.

Table 8-1 XmlDiff Command-Line Options for the C Language

Option	Description
`-e` encoding	Specify default input-file encoding. If no encoding is specified in XML file, this encoding is assumed for input.
`-E` encoding	Specify output/data encoding. DOMs and the `Xdiff` instance document are created in this encoding. Default is 8-bit encoding of Unicode (UTF-8).

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 8-1 (Cont.) XmlDiff Command-Line Options for the C Language

Option	Description
`-h hashLevel`	Specify the hash level. `0` means none. If greater than `1`, starting depth to use hashing for subtrees.
`-g`	Set global optimization (default).
`-l`	Set local optimization.
`-p`	Show this usage help.
`-u`	Disable update operation.

8.2.5 Sample Input Document

A sample input XML document is presented.

Example 8-1 is a sample XML document that you can use to explain updates resulting from using both `XmlDiff` and `XmlPatch`. It is followed by some hypothetical changes.

Assume that there is another file, `book2.xml`, that looks just like Example 8-1 except that it causes these actions:

- Deletes "The Eleventh Commandment", a `delete-node` operation.
- Changes the country code for the "C++ Primer" to US from USA, an `update-node` operation.
- Adds a description to "Emperor's New Mind", an `append-node` operation.
- Adds the edition to "Evening News", an `insert-node-before` operation.
- Updates the price of "Evening News", an `update-node` operation.

Example 8-1 book1.xml

```
<?xml version="1.0"?>
<booklist xmlns="http://booklist.oracle.com">
  <book>
    <title>Twelve Red Herrings</title>
    <author>Jeffrey Archer</author>
    <publisher>Harper Collins</publisher>
    <price>7.99</price>
  </book>
  <book>
    <title language="English">The Eleventh Commandment</title>
    <author>Jeffrey Archer</author>
    <publisher>McGraw Hill</publisher>
    <price>3.99</price>
  </book>
  <book>
    <title language="English" country="USA">C++ Primer</title>
    <author>Lippmann</author>
    <publisher>Harper Collins</publisher>
    <price>4.99</price>
  </book>
  <book>
    <title>Emperor's New Mind</title>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
    <author>Roger Penrose</author>
    <publisher>Oxford Publishing Company</publisher>
    <price>15.9</price>
  </book>
  <book>
    <title>Evening News</title>
    <author>Arthur Hailey</author>
    <publisher>MacMillan Publishers</publisher>
    <price>9.99</price>
  </book>
</booklist>
```

8.2.6 Sample Xdiff Instance Document

A sample `Xdiff` instance document is presented.

This section shows the `Xdiff` instance document produced by the comparison of these two XML files described in the previous section. The sections that follow explain the XML processing instructions and the operations on this document.

You can invoke `XmlDiff`:

```
> xmldiff book1.xml book2.xml
```

You can also examine the sample application for arguments and flags.

Example 8-2 Sample Xdiff Instance Document

```
<?xml version="1.0" encoding="UTF-8"?>
<xd:xdiff xsi:schemaLocation="http://xmlns.oracle.com/xdb/xdiff.xsd
xmlns:xd="http://xmlns.oracle.com/xdb/xdiff.xsd" xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance"
xmlns:oraxdfns_0="http://booklist.oracle.com">
    <?oracle-xmldiff operations-in-docorder="true" output-model="snapshot"
        diff-algorithm="global"?>
    <xd:delete-node xd:node-type="element" xd:xpath="/oraxdfns_0
              :booklist[1]/oraxdfns_0:book[2]"/>
    <xd:update-node xd:node-type="attribute"
         xd:parent-xpath="/oraxdfns_0:booklist[1]/oraxdfns_0:book[3]/
oraxdfns_0
              :title[1]" xd:attr-local="country">
                <xd:content>US</xd:content>
    </xd:update-node>
    <xd:append-node xd:node-type="element" xd:parent-xpath="/oraxdfns_0
              :booklist[1]/oraxdfns_0:book[4]">
        <xd:content>
            <oraxdfns_0:description> This is a classic </
oraxdfns_0:description>
        </xd:content>
    </xd:append-node>
    <xd:insert-node-before xd:node-type="element" xd:xpath="/oraxdfns_0
              :booklist[1]/oraxdfns_0:book[5]/oraxdfns_0:author[1]">
        <xd:content>
            <oraxdfns_0:edition>Hardcover</oraxdfns_0:edition>
        </xd:content>
    </xd:insert-node-before>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
    <xd:update-node xd:node-type="text" xd:xpath="/oraxdfns_0
           :booklist[1]/oraxdfns_0:book[5]/oraxdfns_0:price[1]/text()[1]">
        <xd:content>12.99</xd:content>
    </xd:update-node>
</xd:xdiff>
```

8.2.7 Output Model and XML Processing Instructions

The `Xdiff` instance document uses some XML processing instructions (shown in bold in the previous section) that are used to represent certain aspects of the differencing process.

See Xdiff Schema. These instructions and related options are:

- `operations-in-docorder`: Options are `true` or `false`:
  - `true`—The `Xdiff` instance document refers to the nodes from the first document in the same order as in the document.
  - `false`—The `Xdiff` instance document does not refer to the nodes from the first document in the same order as in the document.

  The output of global optimization meets the `operations-in-docorder` requirement, but local optimization does not.
- `output-model`: Options are:
  - `snapshot`—`Xmldiff` generates output in snapshot model and follows the UNIX *diff* model. Each operation uses `XPath` as if no operations have been applied to the input document. This is the default. `XmlPatch` can handle this model only if `operations-in-docorder` is set to `true` and the `XPaths` are simple. Simple `XPaths` require a child axis, no wild cards, and must use positional predicates, such as `/root[1]/child[2]/text()[2]`.
  - `current`—Each operation uses `XPath` as if all operations up to the previous one have been applied to the input document. Even though `XmlDiff` does not generate differences in the current model, `XmlPatch` can handle a hand-crafted `diff` document in the current model
- `diff-algorithm`: Options indicate which optimization generated the differences.
  - Global optimization
  - Local optimization

Related Topics

- User Options for Comparison Optimization
  There are two optimization options for comparison: global and local optimization.

8.2.8 Xdiff Operations

`XmlDiff` captures differences using operations indicated by the `Xdiff` instance document. The `XmlDiff` operations are described.

Table 8-2 Xdiff Operation Attributes

Attribute	Description
`parent-path` or `xpath`	Specifies the XPATH location of the parent node of the operand node or the XPATH location of node.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 8-2 (Cont.) Xdiff Operation Attributes

Attribute	Description
`node-type`	Specifies the type of the operand node.
`content`	Child element that specifies the new subtree or value appended or inserted.

The `Xdiff` operations, presented in the `Xdiff` instance document, are:

- `append-node`:

  The `append-node` element specifies that a node of the given type is added as the last child of the given parent.

- `insert-node-before`:

  The `insert-node-before` element specifies that a node of the given type is inserted before the given reference node.

- `delete-node`:

  The `delete-node` element specifies that the node be deleted along with all its children. You can use this element to delete elements, comments, and so on.

- `update-node`:

  `update-node` specifies that the value associated with the node with the given `XPath` expression is updated to the new value, which is specified. Content is the value for a text node. The value of an attribute is the value for an attribute node.

  - Update for Text Nodes:
    - Generation of update node operations can be turned off by the user.
    - The value of an attribute is the value for an attribute node.
    - `update-node` is generated for text nodes only by global optimization.
  - Update for Elements:
    - `XmlDiff` does not generate update operations for element nodes.

      You can either manually modify the `Xdiff` instance document to create an update operation that works with `XmlPatch`, or provide a totally hand-written `Xdiff` instance document. All children of the element operated on by the update are deleted. Any new subtree specified under the content node is imported.

8.2.9 Format of Xdiff Instance Document

The output of `XmlDiff`, the `Xdiff` instance document, is an XML document that conforms to the `Xdiff` XML schema. The output document contains a sequence of operations describing the differences between the two input documents. If you apply the differences to the first document, you obtain the second document.

8.2.10 Xdiff Schema

An `Xdiff` XML schema, to which an `Xdiff` instance document (output) adheres, is presented.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Example 8-3 Xdiff Schema: xdiff.xsd

```
<schema targetNamespace="http://xmlns.oracle.com/xdb/xdiff.xsd"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:xd="http://xmlns.oracle.com/xdb/xdiff.xsd"
    version="1.0" elementFormDefault="qualified"
    attributeFormDefault="qualified">
    <annotation>
        <documentation xml:lang="en">
         Defines the structure of XML documents that capture the difference
         between two XML documents. Changes that are not supported by Oracle
         XmlDiff may not be expressible in this schema.

       'oracle-xmldiff' PI in Xdiff document:

       We use 'oracle-xmldiff' PI to describe certain aspects of the diff.
       The PI denotes values for 'operations-in-docorder' and 'output-model'.
       The output of XmlDiff has the PI always. If the user hand-codes a diff
doc
       then it must also have the PI in it as the first child of top level
xdiff
       element, to be able to call XmlPatch.

       operations-in-docorder:
       Can be either 'true' or 'false'.
       If true, the operations in the diff document refer to the
       elements of the input doc in the same order as document order. Output
of
       global algorithm meets this requirement while local does not.

       output-model:
       output models for representing the diff. Can be either 'Snapshot' or
       'Current'.

       Snapshot model:
       Each operation uses Xpaths as if no operations
       have been applied to the input document. (like UNIX diff)
       This is the model used in the output of XmlDiff. XmlPatch works with
       this (and the current model too).
       For XmlPatch to handle this model, "operations-in-docorder" must be
       true and the Xpaths must be simple. (see XmlDif C API documentation).

       Current model:
       Each operation uses Xpaths as if all operations till the previous one
       have been applied to the input document. Works with XmlPatch even if
       the 'operations-in-docorder' criterion is not met and the xpaths are
       not simple.
       <!-- Example:
            <?oracle-xmldiff operations-in-docorder="true" output-model=
            "snapshot" diff-algorithm="global"?>
        -->
      </documentation>
    </annotation>
    <!-- Enumerate the supported node types -->
    <simpleType name="xdiff-nodetype">
        <restriction base="string">
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
                    <enumeration value="element"/>
                    <enumeration value="attribute"/>
                    <enumeration value="text"/>
                    <enumeration value="cdata"/>
                    <enumeration value="entity-reference"/>
                    <enumeration value="entity"/>
                    <enumeration value="processing-instruction"/>
                    <enumeration value="notation"/>
                    <enumeration value="comment"/>
                 </restriction>
            </simpleType>

            <element name="xdiff">
                <complexType>
                    <choice minOccurs="0" maxOccurs="unbounded">
                        <element name="append-node">
                            <complexType>
                                <sequence>
                                    <element name="content" type="anyType"/>
                                </sequence>
                                <attribute name="node-type" type="xd:xdiff-
nodetype"/>
                                <attribute name="xpath" type="string"/>
                                <attribute name="parent-xpath" type="string"/>
                                <attribute name="attr-local" type="string"/>
                                <attribute name="attr-nsuri" type="string"/>
                            </complexType>
                        </element>

                        <element name="insert-node-before">
                            <complexType>
                                <sequence>
                                    <element name="content" type="anyType"/>
                                </sequence>
                                <attribute name="xpath" type="string"/>
                                <attribute name="node-type" type="xd:xdiff-nodetype"/>

                            </complexType>
                        </element>

                        <element name="delete-node">
                            <complexType>
                                <attribute name="node-type" type="xd:xdiff-nodetype"/>
                                <attribute name="xpath" type="string"/>
                                <attribute name="parent-xpath" type="string"/>
                                <attribute name="attr-local" type="string"/>
                                <attribute name="attr-nsuri" type="string"/>
                            </complexType>
                        </element>
                         <element name="update-node">
                            <complexType>
                                <sequence>
                                    <element name="content" type="anyType"/>
                                </sequence>
                                <attribute name="node-type" type="xd:xdiff-
nodetype"/>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
                        <attribute name="parent-xpath" type="string"/>
                        <attribute name="xpath" type="string"/>
                        <attribute name="attr-local" type="string"/>
                        <attribute name="attr-nsuri" type="string"/>
                    </complexType>
                </element>
                <element name="rename-node">
                    <complexType>
                        <sequence>
                            <element name="content" type="anyType"/>
                        </sequence>
                        <attribute name="xpath" type="string"/>
                        <attribute name="node-type" type="xd:xdiff-
nodetype"/>
                    </complexType>
                </element>
            </choice>
         <attribute name="xdiff-version" type="string"/>
        </complexType>
    </element>
</schema>
```

8.2.11 Using XMLDiff in an Application

In an application, `XmlDiff` takes the source types and locations of the input documents as arguments. The source type can be a URL, file, `orastream` and `stream` context pointers, buffer, and `buffer_length` pointers or the pointer to a DOM document element (`docelement`).

`XmlDiff` returns the document node for the DOM for the `Xdiff` instance document.

XmlDiff builds the DOM for the two documents, if they are not already provided as DOM, before performing a comparison.

> **See Also**
>
> *Oracle AI Database XML C API Reference* for the C API for the flags that control the behavior of `XmlDiff`

Example 8-4 XMLDiff Application

```
# include <xmldf.h>
...
xmlctx     *xctx;
xmldocnode *doc1, *doc2, *doc3;
uword       hash_level;
oratext    *s, *inp1 = "book1.xml", *inp2="book2.xml";
xmlerr      err;
ub4         flags;

flags = 0; /* defaults : global algorithm */
hash_level = 0; /* no hashing */
/* create XML meta context */
if (!(xctx = XmlCreate(&err, (oratext *) "XmlDiff", NULL)))
{
   printf("Failed to create XML context, error %u\n",
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
(unsigned) err);
err_exit("Exiting");
}
/* Load the two input files */
if (!(doc1 = XmlLoadDom(xctx, &err, "file", inp1, "discard_whitespace", TRUE,
 NULL)))
{
   printf("Parsing first file failed, error %u\n", (unsigned)err);
   err_exit((oratext *)"Exiting.");
}
if (!(doc2 = XmlLoadDom(xctx, &err, "file", inp2, "discard_whitespace", TRUE,
    NULL)))
{
   printf("Parsing second file failed, error %u\n", (unsigned)err);
   err_exit((oratext *)"Exiting.");
}

/* run XmlDiff on the DOM trees. */

doc3 = XmlDiff(xctx, &err, flags, XMLDF_SRCT_DOM, doc1, NULL, XMLDF_SRCT_DOM,
               doc2, NULL,hash_level, NULL);

if(!doc3)
   printf("XmlDiff Failed, error %u\n", (unsigned)err);
else
{
if(err != XMLERR_OK)
printf("XmlDiff returned error %u\n", (unsigned)err);
/* Now we have the DOM tree in doc3 which represent the Diff */
...
}

XmlFreeDocument(xctx, doc1);
XmlFreeDocument(xctx, doc2);
XmlFreeDocument(xctx, doc3);
XmlDestroy(xctx);
```

8.2.12 Customized Output

A customized output builder stores differences in any format suitable to the application. You can create your own customized output builder, rather than using the default `Xdiff` instance document, which is generated by `XmlDiff` and that conforms to the `Xdiff` schema.

To create a customized output builder, you must provide a callback that can be called after `XmlDiff` determines the differences. The differences are passed to the callback as an array of `xmdlfop`. The callback may be called multiple times as the differences are being generated.

Using a customized output builder may perform better than using the default, because it does not have to maintain the internal state necessary for `XPath` generation.

By default, `XmlDiff` captures the differences in XML conforming to the `Xdiff` schema. If necessary, plug in your own output builder. The differences are represented as an array `xmldfop`. You must write an output builder callback function. The function signature is:

```
xmlerr(*xdfobcb)(void *uctx, xmldfop *escript, ub4 escript_siz);
```

`uctx` is the user specific context.

`escript` is the array of size `escript_siz`:

```
diff[escript_siz]
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

`mctx` is the memory context.

Supply this memory context through properties to `XmlDiff()`. Use this memory context to allocate `escript`. You must later free `escript`.

Invoke the output builder callback after the differences have been found which happens even before the invocation of `XmlDiff()` returns. The output builder callback can be called multiple times.

Example 8-5 Customized XMLDiff Output

```
/* Sample useage:  */
...
#include <orastruc.h> / * for 'oraprop' * /
...
static oraprop diff_props[] = {
   ORAPROP(XMLDF_PROPN_CUSTOM_OB, XMLDF_PROPI_CUSTOM_OB, POINTER),
   ORAPROP(XMLDF_PROPN_CUSTOM_OBMCX, XMLDF_PROPI_CUSTOM_OBMCX, POINTER),
   ORAPROP(XMLDF_PROPN_CUSTOM_OBUCX, XMLDF_PROPI_CUSTOM_OBUCX, POINTER),
       { NULL }
};
...
oramemctx *mymemctx;
...
xmlerr myob(void *uctx,  xmldfop *escript, ub4 escript_siz)
{
   /* process diff which is available in escript * /

   /* free escript - the caller has to do this * /
   OraMemFree(mymemctx, escript);
}

main()
{
 ...
    myctxt  *myctx;

    diff_props[0].value_oraprop.p_oraprop_v = myob;
    diff_props[1].value_oraprop.p_oraprop_v = mymemctx;
    diff_props[2].value_oraprop.p_oraprop_v = myctx;
    XmlDiff(xctx, &err, 0, doc1, NULL, 0, doc2, NULL, 0, diff_props);
...
```

8.3 Using XmlPatch

`XmlPatch` takes an `Xdiff` instance document, generated by `XmlDiff` or created by another mechanism, and follows the instructions in the `Xdiff` instance document to modify other XML documents.

Copyright © 1999, 2025, Oracle and/or its affiliates.

8.3.1 Using the XmlPatch Command-Line Utility

Command-line options for utility `XmlPatch` are described.

Table 8-3 XmlPatch for C Command-Line Options

Option	Description
`-e` encoding	Specify default input-file encoding. If no encoding is specified in XML file, this encoding is assumed for input.
`-E` encoding	Specify output/data encoding. DOMs and patched document are created in this encoding. Default is UTF-8.
`-i`	Interpret file names as URLs.
`-h`	Show this usage help.

8.3.2 Using XmlPatch in an Application

`XmlPatch` takes the source types and locations of the input document and the `diff` document as arguments. The source type can be a URL, file, `orastream` and `stream` context pointers, buffer and `buffer_length` pointers, or the pointer to a DOM document element (`docelement`).

> **See Also**
>
> *Oracle AI Database XML C API Reference* for the C API for the flags that control the behavior of `XmlPatch`

The modes that were set by the `Xdiff` schema affect how `XmlPatch` works.

If the `output-model` is `Snapshot`, `XmlPatch` only works if `operations-in-docorder` is `TRUE`.

If the `output-model` is `Current`, it is not necessary that `operations-in-docorder` be set to `TRUE`.

Example 8-6 Sample Application for XmlPatch

```
...
#include <xmldf.h>
...
xmlctx     *xctx;
xmldocnode *doc1, *doc2;
oratext    *s;
oratext    *inp1 = "book1.xml"; /* input document */
oratext    *inp2 = "diff.xml", /* diff document */
xmlerr      err;

/* create XML meta context */
if (!(xctx = XmlCreate(&err, (oratext *) "XmlPatch", NULL)))
{
   printf("Failed to create XML context, error %u\n",
(unsigned) err);
err_exit("Exiting");
}
/* Load the two input files */
if (!(doc1 = XmlLoadDom(xctx, &err, "file", inp1, "discard_whitespace", TRUE,
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
    NULL)))
{
   printf("Parsing first file failed, error %u\n", (unsigned)err);
   err_exit((oratext *)"Exiting.");
}
if (!(doc2 = XmlLoadDom(xctx, &err, "file", inp2, "discard_whitespace", TRUE,
    NULL)))
{
   printf("Parsing second file failed, error %u\n", (unsigned)err);
   err_exit((oratext *)"Exiting.");
}

/* call XmlPatch */
if(!XmlPatch(xctx, &err, 0, XMLDF_SRCT_DOM, doc1, NULL, XMLDF_SRCT_DOM,
           doc2, NULL, NULL));

   printf("XmlPatch Failed, error %u\n", (unsigned)err);
else
{
if(err != XMLERR_OK)
printf("XmlPatch returned error %u\n", (unsigned)err);
/* Now we have the patched document in doc1 */
...
}

XmlFreeDocument(xctx, doc1);
XmlFreeDocument(xctx, doc2);
XmlDestroy(xctx);
```

8.4 Using XmlHash

`XmlHash` computes a hash value for an XML tree. If the hash values of two trees are equal, it is probable that they are the same XML. You can use `XmlHash` to do a quick comparison to see if an XML tree is already in the database.

You can run `XmlDiff` again, if necessary, on any matches, to be absolutely certain there is a match. You can compute the hash value of the new document and query the database for it.

Example 8-7 shows a sample program that uses `XmlHash`.

Example 8-7 XmlHash Program

```
sword main(sword argc, char *argv[])
{
   xmlctx     *xctx;
   xmldfsrct   srct;
   oratext    *data_encoding, *input_encoding, *s, *inp1;
   ub1         flags;
   xmlerr      err;
   ub4         num_args;
   xmlhasht    digest;
   flags = 0; /* defaults */
   srct = XMLDF_SRCT_FILE;
   inp1 = "somexml.xml";
   xctx = XmlCreate(&err, (oratext *) "XmlHash", NULL);

   if (!xctx)
   {
      /* handle error with creating xml context and exit */
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
      ...
   }

   /* run XmlHash */
   err = XmlHash(xctx, &digest, 0, srct, inp1, NULL, NULL);
   if(err)
      printf("XmlHash returned error:%d \n", err);
   else
      txdfha_pd(digest);

   XmlDestroy(xctx);

   return (sword )err;
}

/* print bytes in xml hash */
static void txdfha_pd(xmlhasht digest)
{
   ub4  i;

   for(i = 0; i < digest.l_xmlhasht; i++)
      printf("%x ", digest.d_xmlhasht[i]);

   printf("\n");
}
```

8.4.1 Invoking XmlDiff and XmlPatch

`XmlDiff` and `XmlPatch` can be called as command-line tools and from the C language. They are also available as SQL functions.

> **See Also**
>
> - *Oracle AI Database SQL Language Reference* `XMLDiff`
> - *Oracle AI Database SQL Language Reference*, `XMLPatch`

Copyright © 1999, 2025, Oracle and/or its affiliates.

9

Using SOAP with the Oracle XML Developer's Kit for C

An explanation is given of how to use Simple Object Access Protocol (SOAP) with the Oracle XML Developer's Kit (XDK) for C.

> **See Also**
>
> *Oracle XML DB Developer's Guide*

9.1 Introduction to SOAP for C

SOAP is an Extensible Markup Language (XML) protocol for exchanging structured and typed information between peers using HTTP and HTTPS in a distributed environment. Only HTTP 1.0 is supported in XDK for Oracle Database 10g Release 2.

SOAP has three parts:

- The SOAP envelope which defines how to present what is in the message, who must process the message, and whether that processing is optional or mandatory.
- A set of serialization and deserialization rules for converting application data types to and from XML.
- A SOAP remote procedure call (RPC) that defines calls and responses.

> **Note**
>
> RPC and serialization/deserialization are not supported in this release.

SOAP is operating system and language-independent because it is XML-based. This chapter presents the C implementation of the functions that read and write the SOAP message.

SOAP Version 1.2 is the definition of an XML-based message which is specified as an XML Infoset (an abstract data set, it could be XML 1.0) that gives a description of the message contents. Version 1.1 is also supported.

9.1.1 SOAP Messaging Overview

SOAP is a lightweight protocol for sending and receiving requests and responses across the Internet. Because it is based on XML and transport protocols such as HTTP, it is not blocked by most firewalls. SOAP is independent of operating system, implementation language, and object model.

The power of SOAP is its ability to act as the glue between heterogeneous software components. For example, Visual Basic clients can invoke Common Object Request Broker

Copyright © 1999, 2025, Oracle and/or its affiliates.

Architecture (CORBA) services running on UNIX computers; Macintosh clients can invoke Perl objects running on Linux.

SOAP messages have these parts:

- An **envelope** that contains the message, defines how to process the message and who processes it, and whether processing is optional or mandatory. The `Envelope` element is required.
- A set of **encoding rules** that describe the data types for the application. These rules define a serialization mechanism that converts the application data types to and from XML.
- A **remote procedure call (RPC)** request and response convention. This required element is called a body element. The `Body` element contains a first subelement whose name is the name of a method. This method request element contains elements for each input and output parameter. The element names are the parameter names. RPC is not currently supported in this release.

SOAP is independent of any transport protocol. Nevertheless, SOAP used over HTTP for remote service invocation has emerged as a standard for delivering programmatic content over the Internet.

Besides being independent of transfer protocol, SOAP is also independent of operating system. In other words, SOAP enables programs to communicate even when they are written in different languages and run on different operating systems.

9.1.1.1 SOAP Message Format

Types of SOAP messages are described.

- Requests for a service, including input parameters
- Responses from the requested service, including return value and output parameters
- Optional fault elements containing error codes and information

In a SOAP message, the **payload** contains the XML-encoded data. The payload contains no processing information. In contrast, the message header may contain processing information.

9.1.1.1.1 SOAP Requests

SOAP requests are described.

In SOAP requests, the XML payload contains several elements that include:

- Root element
- Method element
- Header elements (optional)

Example 9-1 shows the format of a sample SOAP message request. A `GetLastTradePrice` SOAP request is sent to a `StockQuote` service. The request accepts a string parameter representing the company stock symbol and returns a float representing the stock price in the SOAP response.

Example 9-1 SOAP Request Message

```
POST /StockQuote HTTP/1.0
Host: www.stockquoteserver.com
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
<SOAP-ENV:Envelope  xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
            SOAP-ENV:encodingStyle="http://www.w3.org/2003/05/soap-encoding/">
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>ORCL</symbol>
    <m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

In Example 9-1, the XML document is the SOAP message. The `<SOAP-ENV:Envelope>` element is the top-level element of the XML document. The payload is represented by the method element `<m:GetLastTradePrice>`. XML namespaces distinguish SOAP identifiers from application-specific identifiers.

The first line of the header specifies that the request uses HTTP as the transport protocol:

```
POST /StockQuote HTTP/1.1
```

Because SOAP is independent of transport protocol, the rules governing XML payload format are independent of the use of HTTP for transport of the payload. This HTTP request points to the URI `/StockQuote`. Because the SOAP specification is silent on the issue of component activation, the code behind this URI determines how to activate the component and invoke the `GetLastTradePrice` method.

9.1.1.1.2 Example of a SOAP Response

An example of a SOAP response is presented.

Example 9-2 shows the format of the response to the request in Example 9-1. Element `<Price>` contains the stock price for `ORCL` requested by the first message.

The messages shown in Example 9-1 and Example 9-2 show two-way SOAP messaging, that is, a SOAP request that is answered by a SOAP response. A one-way SOAP message does not require a SOAP message in response.

Example 9-2 SOAP Response Message

```
HTTP/1.0 200 OK
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope  xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
 SOAP-ENV:encodingStyle="http://www.w3.org/2003/05/soap-encoding/">
  <SOAP-ENV:Body>
    <m:GetLastTradePriceResponse xmlns:m="Some-URI">
      <Price>13.5</Price>
    </m:GetLastTradePriceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

9.1.2 Using SOAP Clients

SOAP clients are user-written applications that generate XML documents. The documents make a request for a SOAP service and handle a SOAP response. The SOAP implementation in XDK handles requests from any client that sends a valid SOAP request.

The SOAP client application programming interface (API) has these features:

Copyright © 1999, 2025, Oracle and/or its affiliates.

- Supports a synchronous invocation model for requests and responses
- Facilitates the writing of client applications to make SOAP requests
- Encapsulates the creation of the SOAP request and the details of sending the request over the underlying transport protocol
- Supports a pluggable transport, allowing the client to easily change the transport (available transports include HTTP and HTTPS, but only HTTP 1.0 is supported in this release)

A SOAP client must perform these steps to make a request and receive a response:

1. Gather all parameters that are needed to invoke a service.
2. Create a SOAP service request message, which is an XML message that is built according to the SOAP protocol. It contains all the values of all input parameters encoded in XML. This process is called **serialization**.
3. Submit the request to a SOAP server using a transport protocol that is supported by the SOAP server.
4. Receive a SOAP response message.
5. Determine the success or failure of the request by handling the SOAP Fault element.
6. Convert the returned parameter from XML to native data type. This process is called **deserialization**.
7. Use the result as needed.

9.1.3 Using SOAP Servers

The steps performed by a SOAP server when executing a SOAP service request are described.

1. The SOAP server receives the service request.
2. The server parses the XML request and then decides whether to execute or reject the message.
3. If the message is executed, then the server determines whether the requested service exists.
4. The server converts all input parameters from XML into data types that the service understands.
5. The server invokes the service.
6. The server converts the return parameter to XML and generates a SOAP response message.
7. The server sends the response message back to the caller.

9.2 SOAP C Functions

The SOAP C implementation uses the `xml.h` header. A context of type `xmlctx` must be created before a SOAP context can be created.

HTTP aspects of SOAP are hidden from the user. SOAP endpoints are specified as a couple (binding, endpoint) where binding is of type `xmlsoapbind` and the endpoint is a (`void *`) depending on the binding. Currently, only one binding is supported, `XMLSOAP_BIND_HTTP`. For HTTP binding, the endpoint is an (`OraText *`) URL.

Copyright © 1999, 2025, Oracle and/or its affiliates.

The SOAP layer creates and transports SOAP messages between endpoints, and decomposes received SOAP messages.

The C functions are declared in `xmlsoap.h`. Here is the beginning of that header file:

> **See Also**
>
> *Oracle AI Database XML C API Reference* for the C SOAP APIs

Example 9-3 SOAP C Functions Defined in xmlsoap.h

```
FILE NAME
     xmlsoap.h - XML SOAP APIs

FILE DESCRIPTION
     XML SOAP Public APIs

PUBLIC FUNCTIONS
     XmlSoapCreateCtx         - Create and return a SOAP context
     XmlSoapDestroyCtx        - Destroy a SOAP context

     XmlSoapCreateConnection  - Create a SOAP connection object
     XmlSoapDestroyConnection - Destroy a SOAP connection object

     XmlSoapCall              - Send a SOAP message & wait for reply

     XmlSoapCreateMsg         - Create and return an empty SOAP message
     XmlSoapDestroyMsg        - Destroy a SOAP message created
                                   w/XmlSoapCreateMsg

     XmlSoapGetEnvelope       - Return a SOAP message's envelope
     XmlSoapGetHeader         - Return a SOAP message's envelope header
     XmlSoapGetBody           - Return a SOAP message's envelope body

     XmlSoapAddHeaderElement  - Adds an element to a SOAP header
     XmlSoapGetHeaderElement  - Gets an element from a SOAP header

     XmlSoapAddBodyElement    - Adds an element to a SOAP message body
     XmlSoapGetBodyElement    - Gets an element from a SOAP message body

     XmlSoapSetMustUnderstand - Set mustUnderstand attr for SOAP hdr elem
     XmlSoapGetMustUnderstand - Get mustUnderstand attr from SOAP hdr elem

     XmlSoapSetRole           - Set role for SOAP header element
     XmlSoapGetRole           - Get role from SOAP header element

     XmlSoapSetRelay          - Set relay Header element property
     XmlSoapGetRelay          - Get relay Header element property

     XmlSoapSetFault          - Set Fault in SOAP message
     XmlSoapHasFault          - Does SOAP message have a Fault?
     XmlSoapGetFault          - Return Fault code, reason, and details

     XmlSoapAddFaultReason    - Add additional Reason to Fault
     XmlSoapAddFaultSubDetail - Add additional child to Fault Detail
     XmlSoapGetReasonNum      - Get number of Reasons in Fault element
     XmlSoapGetReasonLang     - Get a lang of a reasons with a
                                   particular iindex.
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
        XmlSoapError            - Get error message(s)

*/

#ifndef XMLSOAP_ORACLE
# define XMLSOAP_ORACLE

# ifndef XML_ORACLE
#  include <xml.h>
# endif

/*---------------------------------------------------------------------------
                Package SOAP - Simple Object Access Protocol APIs

     W3C: "SOAP is a lightweight protocol for exchange of information
     in a decentralized, distributed environment.  It is an XML based
     protocol that consists of three parts: an envelope that defines a
     framework for describing what is in a message and how to process
     it, a set of encoding rules for expressing instances of
     application-defined datatypes, and a convention for representing
     remote procedure calls and responses."
     Atachments are allowed only in Soap 1.1
     In Soap 1.2 body may not have other elements if Fault is present.

     Structure of a SOAP message:

     [SOAP message (XML document)
        [SOAP envelope
             [SOAP header?
                 element*
             ]
             [SOAP body
                 (element* | Fault)?
             ]
        ]
     ]
---------------------------------------------------------------------------*/
...
```

9.3 SOAP Example 1: Sending an XML Document

An XML document is presented that shows a request to a travel company for a reservation on a plane flight from New York to Los Angeles for John Smith. A simple example creates the XML document, sends it, receives and decomposes a reply. There is some minimal error checking.

The `DEBUG` option is shown for correcting anomalies. The program may not work on all operating systems. To send this XML document, the first client C program follows these steps:

1. After declaring variables in `main()`, an XML context, `xctx`, is created using `XmlCreate()` and the context is then used to create a SOAP context, `ctx`, using `XmlSoapCreateCtx()`.
2. To construct the message, `XmlSoapCreateMsg()` is called and returns an empty SOAP message.
3. The header is constructed using `XmlSoapAddHeaderElement()`, `XmlSoapSetRole()`, `XmlSoapSetMustUnderstand()`, and `XmlDomAddTextElem()` to fill in the envelope with text.
4. The body elements are created by `XmlSoapAddBodyElement()`, `XmlDomCreateElemNS()`, and a series of invocations of `XmlDomAddTextElem()`. Then `XmlDomAppendChild()` completes the section of the body specifying the New York to Los Angeles flight.

Copyright © 1999, 2025, Oracle and/or its affiliates.

5. The return flight is built in an analogous way. The lodging is added with another `XmlSoapAddBodyElement()` invocation.
6. The connection must be created next with `XmlSoapCreateConnection()`, specifying HTTP binding (the only binding available now) and an endpoint URL.
7. The function `XmlSoapCall()` sends the message over the defined connection with the SOAP server, and then waits for the reply.
8. The message reply is returned in the form of another SOAP message. This is done with `XmlSaveDom()` and `XmlSoapHasFault()` used with `XmlSoapGetFault()` to check for a fault and analyze the fault. The fault is parsed into its parts, which is output in this example.
9. If there was no fault returned, this is followed by `XmlSoapGetBody()` to return the envelope body. `XmlSaveDom()` completes the analysis of the returned message.
10. To clean up, use `XmlSoapDestroyMsg()` on the message and on the reply, `XmlDestroyCtx()` to destroy the SOAP context, and `XmlDestroy()` to destroy the XML context.

Example 9-4 Example 1 SOAP Message

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
          env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
           env:mustUnderstand="true">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
   <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/employees"
          env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
           env:mustUnderstand="true">
   <n:name>John Smith</n:name>
  </n:passenger>
 </env:Header>
 <env:Body>
  <p:itinerary
    xmlns:p="http://travelcompany.example.org/reservation/travel">
   <p:departure>
     <p:departing>New York</p:departing>
     <p:arriving>Los Angeles</p:arriving>
     <p:departureDate>2001-12-14</p:departureDate>
     <p:departureTime>late afternoon</p:departureTime>
     <p:seatPreference>aisle</p:seatPreference>
   </p:departure>
   <p:return>
     <p:departing>Los Angeles</p:departing>
     <p:arriving>New York</p:arriving>
     <p:departureDate>2001-12-20</p:departureDate>
     <p:departureTime>mid-morning</p:departureTime>
     <p:seatPreference/>
   </p:return>
  </p:itinerary>
  <q:lodging
   xmlns:q="http://travelcompany.example.org/reservation/hotels">
   <q:preference>none</q:preference>
  </q:lodging>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
 </env:Body>
</env:Envelope>
```

Example 9-5 Example 1 SOAP C Client

```
#ifndef S_ORACLE
# include <s.h>
#endif

#ifndef XML_ORACLE
# include <xml.h>
#endif

#ifndef XMLSOAP_ORACLE
# include <xmlsoap.h>
#endif


#define MY_URL "http://my_url.com"

/* static function declaration */
static xmlerr add_ns_decl(xmlsoapctx  *ctx, xmlctx *xctx, xmlelemnode *elem,
                        oratext *pfx, oratext *uri);


sb4 main( sword argc, char *argv[])
{
    xmlctx      *xctx;
    xmlerr       xerr;
    xmlsoapctx  *ctx;
    oratext     *url;
    xmlsoapcon  *con;

    xmldocnode  *msg1, *reply, *msg2, *msg3;
    xmlelemnode *res, *pas, *pref, *itin, *departure, *ret, *lodging;
    xmlelemnode *departing, *arriving, *trans, *text, *charge, *card, *name;
    xmlelemnode *body, *header;
    boolean      has_fault;
    oratext     *code, *reason, *lang, *node, *role;
    xmlelemnode *detail;
    oratext *comp_uri   = "http://travelcompany.example.org/";
    oratext *mres_uri   = "http://travelcompany.example.org/reservation";
    oratext *trav_uri   = "http://travelcompany.example.org/reservation/
travel";
    oratext *hotel_uri  = "http://travelcompany.example.org/reservation/
hotels";
    oratext *npas_uri   = "http://mycompany.example.com/employees";

    oratext *tparty_uri = "http://thirdparty.example.org/transaction";
    oratext *estyle_uri = "http://example.com/encoding";
    oratext *soap_style_uri = "http://www.w3.org/2003/05/soap-encoding";
    oratext *estyle     = "env:encodingStyle";
    oratext *finance_uri = "http://mycompany.example.com/financial";
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
    if (!(xctx = XmlCreate(&xerr, (oratext *)"SOAP_test",NULL)))
    {
        printf("Failed to create XML context, error %u\n", (unsigned) xerr);
        return EX_FAIL;
    }
    /* Create SOAP context */
    if (!(ctx = XmlSoapCreateCtx(xctx, &xerr, (oratext *) "example", NULL)))
       {
         printf("Failed to create SOAP context, error %u\n", (unsigned) xerr);
         return EX_FAIL;
       }


    /* EXAMPLE 1 */
    /* construct message */
    if (!(msg1 = XmlSoapCreateMsg(ctx, &xerr)))
    {
      printf("Failed to create SOAP message, error %u\n", (unsigned) xerr);
      return xerr;
    }
    res = XmlSoapAddHeaderElement(ctx, msg1, "m:reservation", mres_uri,
&xerr);
    xerr = XmlSoapSetRole(ctx, res, XMLSOAP_ROLE_NEXT);
    xerr = XmlSoapSetMustUnderstand(ctx, res, TRUE);
    (void) XmlDomAddTextElem(xctx, res, mres_uri, "m:reference",
                "uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d");
    (void) XmlDomAddTextElem(xctx, res, mres_uri, "m:dateAndTime",
                               "2001-11-29T13:20:00.000-05:00");
    pas = XmlSoapAddHeaderElement(ctx, msg1, "n:passenger", npas_uri, &xerr);
    xerr = XmlSoapSetRole(ctx, pas, XMLSOAP_ROLE_NEXT);
    xerr = XmlSoapSetMustUnderstand(ctx, pas, TRUE);
    (void) XmlDomAddTextElem(xctx, pas, npas_uri, "n:name",
                               "John Smith");
    /* Fill body */
    /* Itinerary */
    itin = XmlSoapAddBodyElement(ctx, msg1, "p:itinerary", trav_uri, &xerr);
    /* Departure */
    departure = XmlDomCreateElemNS(xctx, msg1,  trav_uri, "p:departure");
    (void) XmlDomAddTextElem(xctx, departure, trav_uri,
                               "p:departing","New York");
    (void) XmlDomAddTextElem(xctx, departure, trav_uri,
                               "p:arriving", "Los Angeles");
    (void) XmlDomAddTextElem(xctx, departure, trav_uri,
                               "p:departureDate", "2001-12-14");
    (void) XmlDomAddTextElem(xctx, departure, trav_uri,
                               "p:departureTime", "late afternoon");
    (void) XmlDomAddTextElem(xctx, departure, trav_uri,
                               "p:seatPreference", "aisle");
    XmlDomAppendChild(xctx, itin, departure);

    /* Return */
    ret = XmlDomCreateElemNS(xctx, msg1,  trav_uri, "p:return");
    (void) XmlDomAddTextElem(xctx, ret, trav_uri,
                               "p:departing", "Los Angeles");
    (void) XmlDomAddTextElem(xctx, ret, trav_uri,
                               "p:arriving", "New York");
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
    (void) XmlDomAddTextElem(xctx, ret, trav_uri,
                             "p:departureDate", "2001-12-20");
    (void) XmlDomAddTextElem(xctx, ret, trav_uri,
                             "p:departureTime", "mid-morning");
    pref = XmlDomCreateElemNS(xctx, msg1, trav_uri, "p:seatPreference");
    (void) XmlDomAppendChild(xctx, ret, pref);
    XmlDomAppendChild(xctx, itin, ret);

    /* Lodging */
    lodging = XmlSoapAddBodyElement(ctx, msg1, "q:lodging", hotel_uri, &xerr);
    (void) XmlDomAddTextElem(xctx, lodging, hotel_uri,
                             "q:preference", "none");

#ifdef DEBUG
    /* dump the message in debug mode */
    printf("Message:\n");
    XmlSaveDom(xctx, &xerr, msg1, "stdio", stdout, "indent_step", 1, NULL);
#endif

/* END OF EXAMPLE 1 */

    /* create connection */
    url = MY_URL;
    if (!(con = XmlSoapCreateConnection(ctx, &xerr, XMLSOAP_BIND_HTTP,
          url, NULL, 0, NULL, 0,
          "XTest: baz", NULL)))
      {
        printf("Failed to create SOAP connection, error %u\n", (unsigned)
xerr);
        return xerr;
    }

    reply = XmlSoapCall(ctx, con, msg1, &xerr);
    XmlSoapDestroyConnection(ctx, con);

    if (!reply)
    {
      printf("Call failed, no message returned.\n");
      return xerr;
    }

#ifdef DEBUG
    printf("Reply:\n");
    XmlSaveDom(xctx, &xerr, reply, "stdio", stdout, NULL);
#endif

     printf("\n==== Header:\n ");
    header = XmlSoapGetHeader(ctx, reply, &xerr);
    if (!header)
    {
        printf("NULL\n");
    }
    else
        XmlSaveDom(xctx, &xerr, header, "stdio", stdout, NULL);
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
/* check for fault */
 has_fault = XmlSoapHasFault(ctx, reply, &xerr);
 if(has_fault)
 {
     lang = NULL;
     xerr = XmlSoapGetFault(ctx, reply, &code, &reason, &lang,
                            &node, &role, &detail);
     if (xerr)
     {
         printf("error getting Fault %d\n", xerr);
         return EX_FAIL;
     }
      if(code)
         printf("   Code -- %s\n", code);
     else
         printf("   NO Code\n");
     if(reason)
         printf("   Reason -- %s\n", reason);
     else
         printf("   NO Reason\n");
     if(lang)
         printf("   Lang -- %s\n", lang);
     else
         printf("   NO Lang\n");
     if(node)
         printf("   Node -- %s\n", node);
     else
         printf("   NO Node\n");
     if(role)
         printf("   Role -- %s\n", role);
     else
         printf("   NO Role\n");
     if(detail)
     {
         printf("   Detail\n");
         XmlSaveDom(xctx, &xerr, detail, "stdio", stdout, NULL);
         printf("\n");
     }
     else
         printf("   NO Detail\n");

 }
 else
 {
     body = XmlSoapGetBody(ctx, reply, &xerr);
     printf("==== Body:\n ");
     if (!body)
     {
         printf("NULL\n");
         return EX_FAIL;
     }
     XmlSaveDom(xctx, &xerr, body, "stdio", stdout, NULL);
 }
 (void) XmlSoapDestroyMsg(ctx, reply);
 (void) XmlSoapDestroyMsg(ctx, msg1);
 (void) XmlSoapDestroyCtx(ctx);
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
    XmlDestroy(xctx);
}
```

9.4 SOAP Example 2: A Response Asking for Clarification

A travel company wants to know which New York airport a traveller, John Smith, will depart from: JFK, EWR, or LGA. It sends a response message that asks for such clarification.

To send this XML document as a SOAP message, substitute this code block for the lines beginning with `/* EXAMPLE 1 */` and ending with `/* END OF EXAMPLE 1 */` in Example 9-5

Example 9-6 Example 2 SOAP Message

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
           env:mustUnderstand="true">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
   <m:dateAndTime>2001-11-29T13:35:00.000-05:00</m:dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/employees"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
           env:mustUnderstand="true">
   <n:name>John Smith</n:name>
  </n:passenger>
 </env:Header>
 <env:Body>
  <p:itineraryClarification
    xmlns:p="http://travelcompany.example.org/reservation/travel">
   <p:departure>
     <p:departing>
       <p:airportChoices>
          JFK LGA EWR
       </p:airportChoices>
     </p:departing>
   </p:departure>
   <p:return>
     <p:arriving>
       <p:airportChoices>
         JFK LGA EWR
       </p:airportChoices>
     </p:arriving>
   </p:return>
  </p:itineraryClarification>
 </env:Body>
</env:Envelope>
```

Example 9-7 Example 2 SOAP C Client

```
#define XMLSOAP_MAX_NAME        1024

/* we need this function for examples 2 and 3 */
static xmlerr add_ns_decl(xmlsoapctx  *ctx, xmlctx *xctx, xmlelemnode *elem,
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
                            oratext *pfx, oratext *uri)
{
    oratext     *aq, aqbuf[XMLSOAP_MAX_NAME];
    xmldocnode  *doc;
    oratext     *xmlns = "xmlns:";

    /* if no room for "xmlns:usersprefix\0" then fail now */
    if ((strlen((char *)pfx) + strlen((char *)xmlns)) >
        sizeof(aqbuf))
        return EX_FAIL;
    (void) strcpy((char *)aqbuf, (char *)xmlns);
    strcat((char *)aqbuf, (char *)pfx);
    doc = XmlDomGetOwnerDocument(xctx, elem);
    aq = XmlDomSaveString(xctx, doc, aqbuf);
    XmlDomSetAttrNS(xctx, elem, uri, aq, uri);
    return XMLERR_OK;
}

    /* EXAMPLE 2 */
    /* construct message */
    if (!(msg2 = XmlSoapCreateMsg(ctx, &xerr)))
    {
         printf("Failed to create SOAP message, error %u\n", (unsigned) xerr);
         return xerr;
    }
    res = XmlSoapAddHeaderElement(ctx, msg2, "m:reservation", mres_uri,
&xerr);
    xerr = XmlSoapSetRole(ctx, res, XMLSOAP_ROLE_NEXT);
    xerr = XmlSoapSetMustUnderstand(ctx, res, TRUE);
    (void) XmlDomAddTextElem(xctx, res, mres_uri, "m:reference",
                 "uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d");
    (void) XmlDomAddTextElem(xctx, res, mres_uri, "m:dateAndTime",
                               "2001-11-29T13:35:00.000-05:00");
    pas = XmlSoapAddHeaderElement(ctx, msg2, "n:passenger", npas_uri, &xerr);
    xerr = XmlSoapSetRole(ctx, pas, XMLSOAP_ROLE_NEXT);
    xerr = XmlSoapSetMustUnderstand(ctx, pas, TRUE);
    (void) XmlDomAddTextElem(xctx, pas, npas_uri, "n:name",
                              "John Smith");
    /* Fill body */
    /* Itinerary */
    itin = XmlSoapAddBodyElement(ctx, msg2, "p:itineraryClarification",
                                 trav_uri, &xerr);
    /* Departure */
    departure = XmlDomCreateElemNS(xctx, msg2,  trav_uri, "p:departure");
    departing = XmlDomCreateElem(xctx, msg2,  "p:departing");
    (void) XmlDomAddTextElem(xctx, departing, trav_uri,
                              "p:airportChoices", "JFK LGA EWR");
    (void) XmlDomAppendChild(xctx, departure, departing);
    XmlDomAppendChild(xctx, itin, departure);

    /* Return */
    ret = XmlDomCreateElemNS(xctx, msg2,  trav_uri, "p:return");
    arriving = XmlDomCreateElemNS(xctx, msg2,  trav_uri, "p:arriving");
    (void) XmlDomAddTextElem(xctx, arriving, trav_uri,
                              "p:airportChoices", "JFK LGA EWR");
    XmlDomAppendChild(xctx, ret, arriving);
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
    XmlDomAppendChild(xctx, itin, ret);

#ifdef DEBUG
    XmlSaveDom(xctx, &xerr, msg2, "stdio", stdout, "indent_step", 1, NULL);
#endif
```

9.5 SOAP Example 3: Using POST

An example sends credit card information for John Smith as an XML document using method `POST`. `XmlSoapCall()` writes the HTTP header that precedes the XML message in the example.

The C Client includes this code block which is substituted like the second example in Example 9-5:

Example 9-8 Example 3 SOAP Message

```
POST /Reservations HTTP/1.0
Host: travelcompany.example.org
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
 <env:Header>
   <t:transaction
           xmlns:t="http://thirdparty.example.org/transaction"
           env:encodingStyle="http://example.com/encoding"
           env:mustUnderstand="true" >5</t:transaction>
 </env:Header>
 <env:Body>
  <m:chargeReservation
      env:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
         xmlns:m="http://travelcompany.example.org/">
   <m:reservation xmlns:m="http://travelcompany.example.org/reservation">
    <m:code>FT35ZBQ</m:code>
   </m:reservation>
   <o:creditCard xmlns:o="http://mycompany.example.com/financial">
    <n:name xmlns:n="http://mycompany.example.com/employees">
           John Smith
    </n:name>
    <o:number>123456789099999</o:number>
    <o:expiration>2005-02</o:expiration>
   </o:creditCard>
  </m:chargeReservation>
 </env:Body>
</env:Envelope>
```

Example 9-9 Example 3 SOAP C Client

```
#define XMLSOAP_MAX_NAME        1024

/* we need this function for examples 2 and 3 */
static xmlerr add_ns_decl(xmlsoapctx  *ctx, xmlctx *xctx, xmlelemnode *elem,
                          oratext *pfx, oratext *uri)
{
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
    oratext     *aq, aqbuf[XMLSOAP_MAX_NAME];
    xmldocnode  *doc;
    oratext     *xmlns = "xmlns:";

    /* if no room for "xmlns:usersprefix\0" then fail now */
    if ((strlen((char *)pfx) + strlen((char *)xmlns)) >
        sizeof(aqbuf))
        return EX_FAIL;
    (void) strcpy((char *)aqbuf, (char *)xmlns);
    strcat((char *)aqbuf, (char *)pfx);
    doc = XmlDomGetOwnerDocument(xctx, elem);
    aq = XmlDomSaveString(xctx, doc, aqbuf);
    XmlDomSetAttrNS(xctx, elem, uri, aq, uri);
    return XMLERR_OK;
}

    /* EXAMPLE 3 */
    if (!(msg3 = XmlSoapCreateMsg(ctx, &xerr)))
    {
        printf("Failed to create SOAP message, error %u\n", (unsigned) xerr);
        return xerr;
    }
    trans = XmlSoapAddHeaderElement(ctx,msg3, "t:transaction", tparty_uri,
&xerr);
    xerr = XmlSoapSetMustUnderstand(ctx, trans, TRUE);
    XmlDomSetAttr(xctx, trans, estyle, estyle_uri);
    text = XmlDomCreateText(xctx, msg3, "5");
    XmlDomAppendChild(xctx, trans, text);

    /* Fill body */
    /* Charge Reservation */
    charge =
XmlSoapAddBodyElement(ctx,msg3,"m:chargeReservation",comp_uri,&xerr);
    XmlDomSetAttr(xctx, charge, estyle, soap_style_uri);
    res = XmlDomCreateElemNS(xctx, msg3, mres_uri,  "m:reservation");
    if (add_ns_decl(ctx, xctx, res, "m", mres_uri))
        return EX_FAIL;
    (void) XmlDomAddTextElem(xctx, res, mres_uri,
                             "m:code", "FT35ZBQ");
    (void) XmlDomAppendChild(xctx, charge, res);

    /* create card elem with  namespace */
    card = XmlDomCreateElemNS(xctx, msg3, finance_uri, "o:creditCard");
    if (add_ns_decl(ctx, xctx, card, "o", finance_uri))
        return EX_FAIL;
    name =  XmlDomAddTextElem(xctx, card, npas_uri,
                             "n:name", "John Smith");
    /* add namespace */
    if (add_ns_decl(ctx, xctx, name, "n", npas_uri))
        return EX_FAIL;
    (void)  XmlDomAddTextElem(xctx, card, finance_uri,
                             "o:number", "123456789099999");
    (void) XmlDomAddTextElem(xctx, card, finance_uri,
                             "o:expiration", "2005-02");
    (void) XmlDomAppendChild(xctx, charge, card);
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
#ifdef DEBUG
    XmlSaveDom(xctx, &xerr, msg3, "stdio", stdout, "indent_step", 1, NULL);
#endif
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

Part II
Oracle XML Developer's Kit for Java

This part explains how to use Oracle XML Developer's Kit (XDK) to develop Java applications.

Copyright © 1999, 2025, Oracle and/or its affiliates.

10

Unified Java API for XML

The Unified Java application program interface (API) for Extensible Markup Language (XML) is presented. The APIs that are unified for Oracle XML DB and Oracle XML Developer's Kit (XDK) are described.

10.1 Overview of Unified Java API for XML

With the Unified Java API for XML, you can use the core Java DOM APIs required by both Oracle XML DB and XDK. You can also use the new Java classes that provide extra functionality that is built on top of the Java DOM API.

Unified Java API for XML combines the functionality required by both Oracle XML DB and XDK. Oracle XML DB implements the Java Document Object Model (DOM) API using the Java package `oracle.xdb.dom` and XDK implements it using the `oracle.xml.parser.v2` package.

You can use Unified Java API regardless of where your XML data resides (within the database or outside it), because Unified Java API uses a session pool model of connection management. If you do not specify the connection type as thick (which uses OCI APIs and is C-based) or thin (which uses Java Database Connectivity (JDBC) APIs and is purely Java-based), then a Java DOM API is used to connect to a local document object that resides outside the database.

> **See Also**
>
> *Oracle AI Database XML Java API Reference* for information about the `oracle.xml.parser.v2` package

10.2 Component Unification

Some components that were supported only by the thick connection or only by the thin connection have been unified in Unified Java API for XML.

Those that were supported only by the thin connection and have been unified include the following:

- DOM Parser
- Java API for XML Processing (JAXP) Transformer
- XML SQL Utility (XSU)
- Extensible Stylesheet Language Transformation (XSLT)

10.3 About Moving to the Unified Java API

Unified Java API provides new Java classes that replace the old `oracle.xdb.dom` Java classes. All classes in the `oracle.xdb.dom` package are deprecated. If you are using

Copyright © 1999, 2025, Oracle and/or its affiliates.

deprecated classes, you must migrate to Unified Java API and use `oracle.xml.parser.v2` classes instead.

10.3.1 Java DOM APIs for XMLType Classes

The Java DOM APIs for XMLType classes are listed, together with their deprecated equivalents.

Table 10-1 lists the `oracle.xdb.dom` package classes that were deprecated in Oracle Database 11g Release 1 (11.1) and their Unified Java API for XML equivalents.

Table 10-1 Deprecated XDB Package Classes and Their Unified Java API Equivalents

Deprecated oracle.xdb.dom.* Class	Equivalent oracle.xml.parser.v2 (Unified Java API) Class
`XDBAttribute`	`XMLAttr`
`XDBBinaryDocument`	None
`XDBCData`	`XMLCDATA`
`XDBCharData`	`CharData`
`XDBComment`	`XMLComment`
`XDBDocFragment`	`XMLDocumentFragment`
`XDBDocument`	`XMLDocument`
`XDBDocumentType`	`DTD`
`XDBDOMException`	`XMLDomException`
`XDBDomImplementation`	`XMLDomImplementation`
`XDBDOMNotFoundErrException`	None
`XDBElement`	`XMLElement`
`XDBEntity`	`XMLEntity`
`XDBEntityReference`	`XMLEntityReference`
`XDBNamedNodeMap`	`XMLAttrList`
`XDBNode`	`XMLNode`
`XDBNotation`	`XMLNotation`
`XDBNotImplementedException`	None
`XDBProcInst`	`XMLPI`
`XDBText`	`XMLText`

When you use the Java DOM API to retrieve XML data, you get either an `XMLDocument` instance (if the connection is thin) or an `XDBDocument` instance with method `getDOM()` and an `XMLDocument` instance with method `getDocument()`. Both `XMLDocument` and `XDBDocument` are instances of the World Wide Web Consortium (W3C) DOM interface. The `getDOM()` method and `XDBDocument` class have been deprecated in the Unified Java API for XML.

Table 10-2 lists the deprecated XMLType methods and their Unified Java API equivalents.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 10-2 Deprecated XMLType Methods and Their Unified Java API Equivalents

Deprecated oracle.xdb.XMLType API	Equivalent oracle.xdb.XMLType (Unified Java) API
`getDOM()`	`getDocument()`
`public XMLType createXML(...)`	`public XMLType createXML(..., int kind)` where `kind` is either `XMLDocument.THICK` or `XMLDocument.THIN`

10.3.2 Extension APIs

In addition to the W3C Recommendation, the Unified Java API for XML implementation provides extension APIs that extend the W3C DOM APIs. You can use the Oracle-specific extension APIs for performing basic functions (like connecting to a database) and performance enhancement.

`XMLDocument` is a class that represents the DOM for the instantiated XML document. Retrieve the `XMLType` value from the XML document using the `XMLType` constructor that takes a `Document` argument. For example:

```
XMLType createXML(Connection conn, Document domdoc)
```

To dereference a node manually—that is, to explicitly dereference a document fragment from the DOM tree—use the `freeNode()` extension API in the `oracle.xml.parser.v2` package (`XMLNode` class).

10.3.3 Document Creation Java APIs

A Java API that creates an `XMLDocument` must create either a thin document or a thick document. Because a thick document needs a `Connection` object to establish communication with the database, each document creation API is extended to accept a `Connection` object.

For an `XMLType.createXML` API, you must specify a `Connection` type, which determines the type of object. Old document creation APIs (provided only for backward compatibility), create thin (pure Java) objects unless you specify otherwise.

Table 10-3 lists the `XMLDocument` output, based on `KIND` and `CONNECTION`.

Table 10-3 XMLDocument Output Based on KIND and CONNECTION

XMLDocument.KIND	XMLDocument.CONNECTION	XMLDocument
`XMLDocument.THICK`	Thick or KPRB connection	Thick DOM
`XMLDocument.THICK`	Thin or no connection	Exception
`XMLDocument.THIN`	Any connection type	Thin DOM
Not specified	Any connection type	Non-`XMLType` APIs: Thin DOM `XMLType.createXML` APIs: Determined by connection type—Thick DOM for OCI or KPRB connection, and Thin DOM for Thin connection.

These objects, methods, and classes are available for document creation in the unified Java API:

Copyright © 1999, 2025, Oracle and/or its affiliates.

- `DOMParser` object and `parse()` method

  Use the `DOMParser` object and `parse()` method to parse XML documents. You must specify object type—thick or thin. For a thick object, you must also provide the `Connection` type, using the `DOMParser.setAttribute()` API. For example:

  ```
  DOMParser parser = new oracle.xml.parser.v2.DOMParser();
  parser.setAttribute(XMLDocument.KIND, XMLDocument.THICK);
  parser.setAttribute(XMLDocument.CONNECTION, conn);
  ```

- `DocumentBuilder` object and `DocumentBuilderFactory` class

  Use the `DocumentBuilder` object to parse XML documents using the Java-specific API, JAXP.

  You must create a DOM parser factory with the `DocumentBuilderFactory` class. `DocumentBuilder` builds DOM from input SAX events, taking the Connection from a property set on the `DocumentBuilderFactory`. For example:

  ```
  DocumentBuilderFactory.setAttribute(XMLDocument.CONNECTION, conn);
  DocumentBuilderFactory.setAttribute(XMLDocument.KIND,XMLDocument.THICK);
  ```

  `DocumentBuilderFactory` passes the connection to the `DOMParser` object that creates the document from these APIs:

  ```
  DocumentBuilder.newDocument()
  DocumentBuilder parse(InputStream)
  DocumentBuilder parse(InputStream, int)
  DocumentBuilder.parse(InputSource)
  ```

- XSU methods

  Each XSU method returns an `XMLDocument` to the user. You can specify whether the user wants a thick or thin object. For example:

  ```
  OracleXMLUtil util = new OracleXMLUtil(...);
  util.setAttribute(XMLDocument.KIND, XMLDocument.THICK);
  util.setAttribute(XMLDocument.CONNECTION, conn);
  Document doc = util.getXMLDOMFromStruct(struct, enc);

  OracleXMLQuery query = new OracleXMLQuery(...);
  query.setAttribute(XMLDocument.KIND, XMLDocument.THICK);
  query.setAttribute(XMLDocument.CONNECTION, conn);
  Document doc = query.getXMLDOM (root, meta);

  OracleXMLDocGenDOM dgd = new OracleXMLDocGenDOM(...);
  dgd.setAttribute(XMLDocument.KIND, XMLDocument.THICK);
  dgd.setAttribute(XMLDocument.CONNECTION, conn);
  Document doc = dgd.getXMLDocumentDOM(struct, enc);
  ```

- `XMLType` methods

  You can use the `XMLType.createXML` method to specify the type of the document (thin or thick) that you want the `getDocument()` method to get. In this example, the connection is inferred from `OPAQUE`:

  ```
  XMLType xt = XMLType.createXML(orset.getOPAQUE(1), XMLDocument.THICK);
  Document doc = xt.getDocument();
  ```


Copyright © 1999, 2025, Oracle and/or its affiliates.

> **Note**
>
> You cannot specify the type of an `XMLType` object returned by `ResultSet.getObject()`. The object type is determined by the `Connection` used in the JDBC call that fetches the `ResultSet` from the `XMLType` column.

Copyright © 1999, 2025, Oracle and/or its affiliates.

11

Getting Started with Oracle XML Developer's Kit for Java

How to get started with XDK for Java is described.

11.1 Installing XDK for Java Components

XDK for Java components are included with Oracle Database. This chapter assumes that you installed XDK with Oracle Database and installed the demo programs from the Oracle Database Examples media.

> ⚠ **Caution**
>
> Using the components of Oracle XML Developer's Kit (XDK) to build software programs enables some powerful but potentially dangerous features, such as external entity expansion and recursive expansion. Refer to Security Considerations for Oracle XML Developer's Kit for information about how to use XDK securely.

For a description of the XDK directory structure, see About Installing XDK.

Example 11-1 lists the main directories under the Oracle home directory for Java (This is the UNIX directory structure.) The contents of the subdirectories are listed individually, after the example.

The `bin` directory contains these components:

```
orajaxb
orapipe
oraxml
oraxsl
transx
```

The `lib` directory contains these JAR and ZIP files:

```
classgen.jar
jdev-rt.zip
oraclexsql.jar
transx.zip
xml.jar
xml2.jar
xmldemo.jar
xmlmesg.jar
xmlparserv2.jar
xschema.jar
xsqlserializers.jar
xsu12.jar
```

The `jlib` directory contains these JAR files:

Copyright © 1999, 2025, Oracle and/or its affiliates.

```
orai18n.jar
orai18n-collation.jar
orai18n-mapping.jar
orai18n-utility.jar
```

The `jdbc` directory contains this `lib` subdirectory:

```
| - lib/
     ojdbc6.jar
```

The `rdbms` directory contains this `jlib` subdirectory:

```
| - jlib/
     xdb.jar
```

And, the `xdk` directory contains this `demo` subdirectory:

```
| demo/
   | - java/
        | - classgen/
        | - jaxb/
        | - parser/
        | - pipeline/
        | - schema/
        | - transviewer/
        | - tranxs/
        | - xsql/
        | - xsu/
```

The `/xdk/demo/java` subdirectories contain sample programs and data files for XDK for Java components. The chapters in Oracle XML Developer's Kit for Java explain how to use these programs to learn about the most important Java features.

> **See Also**
>
> Table 1-1 for descriptions of individual XDK for Java components

Example 11-1 Oracle XML Developer's Kit for Java Libraries, Utilities, and Demos

```
- $ORACLE_HOME
    | - bin/
    | - lib/
    | - jlib/
    | - jdbc/
    | - rdbms/
    | - xdk/
```

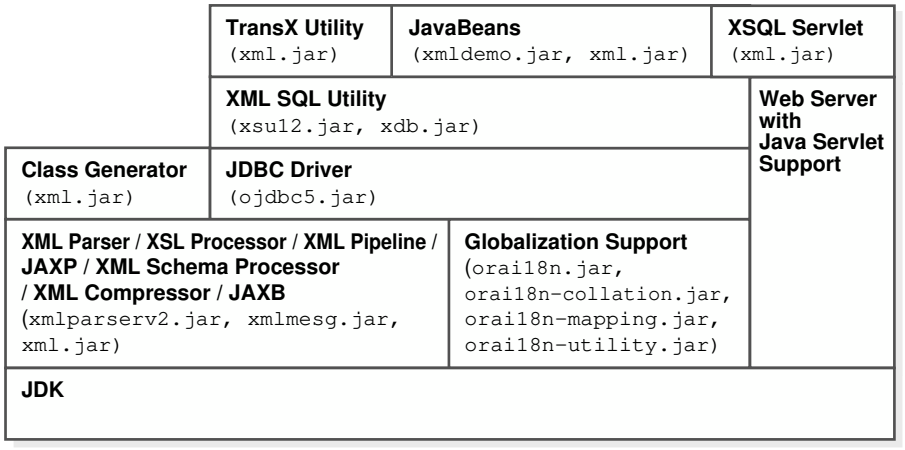
11.2 XDK for Java Component Dependencies

The dependencies of XDK for Java components when using Java Development Kit (JDK) are described.

XDK for Java components are certified and supported with JDK versions 5 and 6. Earlier versions of Java are no longer supported. Figure 11-1 shows the dependencies of XDK for Java components when using JDK 5.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Figure 11-1 Oracle XML Developer's Kit for Java Component Dependencies for JDK 5



XDK for Java components need the libraries in Table 11-1. Some of the libraries are not specific to XDK, but are shared among other Oracle Database components.

Table 11-1 Java Libraries for Oracle XML Developer's Kit for Java Components

Library	Directory	Includes . . .
`classgen.jar`	`$ORACLE_HOME/lib`	Extensible Markup Language (XML) class generator for Java runtime classes. **Note:** This library is maintained only for backward compatibility. Use the Java Architecture for XML Binding (JAXB) class generator in `xml.jar` instead.
`jdev-rt.zip`	`$ORACLE_HOME/lib`	Java graphical user interface (GUI) libraries for use when working with the demos with the Java Development Environment (JDE).
`ojdbc6.jar`	`$ORACLE_HOME/jdbc/lib`	Oracle Java Database Connectivity (JDBC) drivers for Java 6. This Java Archive (JAR) depends on `orai18n.jar` for character set support if you use a multibyte character set other than 8-bit encoding of Unicode (UTF-8), ISO8859-1, or JA16SJIS.
`oraclexsql.jar`	`$ORACLE_HOME/lib`	Most of the XSQL Servlet classes needed to construct XSQL pages. **Note:** This JAR is superseded by `xml.jar` and is retained only for backward compatibility.
`orai18n.jar`	`$ORACLE_HOME/jlib`	Globalization support for JDK 1.2 or above. It is a wrapper of all other Globalization JARs and includes character set converters. If you use a multibyte character set other than UTF-8, ISO8859-1, or JA16SJIS, then put this archive in your `CLASSPATH` so that JDBC can convert the character set of the input file to the database character set when loading XML files with XML SQL Utility (XSU), TransX Utility, or XSQL Servlet.
`orai18n-collation.jar`	`$ORACLE_HOME/jlib`	Globalization collation features: the `OraCollator` class and the `lx3*.glb` and `lx4001[0-9].glb` files.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 11-1 (Cont.) Java Libraries for Oracle XML Developer's Kit for Java Components

Library	Directory	Includes . . .
orai18n-mapping.jar	$ORACLE_HOME/jlib	Globalization locale and character set name mappings: the OraResourceBundle class and lx4000[0-9].glb files. This archive is used mainly by products that need only locale name mapping tables.
orai18n-utility.jar	$ORACLE_HOME/jlib	Globalization locale objects: the OraLocaleInfo class, the OraNumberFormat and OraDateFormat classes, and the lx[01]*.glb files.
transx.zip	$ORACLE_HOME/lib	TransX Utility classes. **Note:** This archive is superseded by xml.jar and is retained only for backward compatibility.
xdb.jar	$ORACLE_HOME/rdbms/jlib	Classes needed by xml.jar to access XMLType, classes needed to access Oracle XML DB Repository, and XMLType Document Object Model (DOM) classes for manipulation of the DOM tree.
xml.jar	$ORACLE_HOME/lib	JAXB and Pipeline Processor classes and classes from these libraries: • oraclexsql.jar • xsqlserializers.jar • transx.jar
xmldemo.jar	$ORACLE_HOME/lib	The visual JavaBeans: XMLTreeView, XMLTransformPanel, XMLSourceView, and DBViewer.
xmlmesg.jar	$ORACLE_HOME/lib	Support for using XML parser with a language other than English.
xmlparserv2.jar	$ORACLE_HOME/lib	Application programming interfaces (APIs) for: • DOM and Simple API for XML (SAX) parsers • XML Schema processor • Extensible Stylesheet Language Transformation (XSLT) processor • XML compression • Java API for XML Processing (JAXP) • Utility functionality such as XMLSAXSerializer and asynchronous DOM Builder This library includes xschema.jar.
xschema.jar	$ORACLE_HOME/lib	XML Schema classes contained in xmlparserv2.jar. **Note:** This JAR file is retained only for backward compatibility.
xsqlserializers.jar	$ORACLE_HOME/lib	Classes that XSQL Servlet needs for serialized output such as PDF. **Note:** This archive is superseded by xml.jar and is retained only for backward compatibility.
xsu12.jar	$ORACLE_HOME/lib	Classes that implement XSU. These classes depend on xdb.jar for XMLType access.

Copyright © 1999, 2025, Oracle and/or its affiliates.

See Also

- *Oracle AI Database Globalization Support Guide* to learn about the Globalization Support libraries
- *Oracle AI Database JDBC Developer's Guide* to learn about the JDBC libraries
- *Oracle XML DB Developer's Guide* to learn about Oracle XML DB

11.3 Setting Up the XDK for Java Environment

You can set up the XDK for Java environment using either an environment variable or a command-line option.

To set up the XDK for Java environment, do either of the following:

- During Oracle Database installation of XDK, manually set the `$CLASSPATH` (UNIX) or `%CLASSPATH%` (Windows) environment variables.
- When compiling and running Java programs at the command line, set the `-classpath` option.

11.3.1 Setting Up XDK for Java Environment Variables for UNIX

The UNIX environment variables needed by XDK for Java components are described.

Table 11-2 UNIX Environment Variables for Oracle XML Developer's Kit for Java Components

Variable	Description
`$CLASSPATH`	Includes: `.:${CLASSPATHJ}:${ORACLE_HOME}/lib/xmlparserv2.jar:${ORACLE_HOME}/lib/xsu12.jar:${ORACLE_HOME}/lib/xml.jar` **Note:** A period (.) to represent the current directory is optional.
`$CLASSPATHJ`	For JDK 5, set: `CLASSPATHJ=${ORACLE_HOME}/jdbc/lib/ojdbc6.jar:${ORACLE_HOME}/jlib/orai18n.jar` Certain character sets need `orai18n.jar`.
`$JAVA_HOME`	Installation directory for the Java JDK, Standard Edition. Modify the path that links to the Java SDK.
`$LD_LIBRARY_PATH`	For OCI JDBC connections: `${ORACLE_HOME}/lib:${LD_LIBRARY_PATH}`
`$PATH`	`${JAVA_HOME}/bin`

After setting up the XDK for Java environment on UNIX, you can use the command-line utilities described in Table 11-3.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 11-3 Oracle XML Developer's Kit for Java UNIX Utilities

Executable/Class	Directory/JAR	Description
`xsql`	`$ORACLE_HOME/bin`	XSQL command-line utility. The script executes the `oracle.xml.xsql.XSQLCommandLine` class. Edit this shell script for your environment before use.
`OracleXML`	`$ORACLE_HOME/lib/xsu12.jar`	XSU command-line utility
`orajaxb`	`$ORACLE_HOME/bin`	JAXB command-line utility
`orapipe`	`$ORACLE_HOME/bin`	Pipeline command-line utility
`oraxml`	`$ORACLE_HOME/bin`	XML parser command-line utility
`oraxsl`	`$ORACLE_HOME/bin`	XSLT processor command-line utility
`transx`	`$ORACLE_HOME/bin`	TransX command-line utility

Related Topics

- Using the XSQL Pages Command-Line Utility
  XDK includes a command-line Java interface that runs the XSQL page processor. You can process any XSQL page with the XSQL command-line utility.
- Using the XSU Command-Line Utility
  XDK includes a command-line Java interface for XSU. XSU command-line options are provided through the Java class `OracleXML`.
- Using the JAXB Class Generator Command-Line Utility
  XDK includes `orajaxb`, which is a command-line Java interface that generates Java classes from input XML schemas. Shell scripts `$ORACLE_HOME/bin/orajaxb` and `%ORACLE_HOME%\bin\orajaxb.bat` execute class `oracle.xml.jaxb.orajaxb`.
- Using the XML Pipeline Processor Command-Line Utility
  The command-line interface for the XML Pipeline processor is named `orapipe`. The Pipeline processor is packaged with Oracle Database. By default, the Oracle Universal Installer installs the utility on disk in `$ORACLE_HOME/bin`.
- Using the Java XML Parser Command-Line Utility (oraxml)
  The `oraxml` utility, which is located in `$ORACLE_HOME/bin` (UNIX) or `%ORACLE_HOME%\bin` (Windows), is a command-line interface that parses XML documents. It checks for both well-formedness and validity.
- Using the XSLT Processor Command-Line Utility
  XDK includes `oraxsl`, which is a command-line Java interface that can apply a stylesheet to multiple XML documents. The `$ORACLE_HOME/bin/oraxsl` and `%ORACLE_HOME%\bin\oraxsl.bat` shell scripts execute the `oracle.xml.jaxb.oraxsl` class.
- Using the TransX Command-Line Utility
  TransX utility is packaged with Oracle Database. By default, the Oracle Universal Installer installs the utility on disk.

11.3.2 Testing the XDK for Java Environment on UNIX

A UNIX shell script is provided to test the XDK Java environment.

If your environment is set up correctly, then the UNIX shell script in Example 11-2 generates version and usage information for the utilities in Table 11-3.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Example 11-2 Testing the Oracle XML Developer's Kit for Java Environment on UNIX

```
#!/usr/bin/tcsh
echo;echo "BEGIN TESTING";echo
echo;echo "now testing the XSQL utility...";echo
xsql
echo; echo "now testing the XSU utility...";echo
java OracleXML
echo;echo "now testing the JAXB utility...";echo
orajaxb -version
echo;echo "now testing the Pipeline utility...";echo
orapipe -version
echo;echo "now testing the XSLT Processor utility...";echo
oraxsl
echo;echo "now testing the TransX utility...";echo
transx
echo;echo "END TESTING"
```

11.3.3 Setting Up XDK for Java Environment Variables for Windows

The Microsoft Windows environment variables needed by XDK for Java components are described.

Table 11-4 describes the Windows environment variables that the XDK for Java components need.

Table 11-4 Windows Environment Variables for Oracle XML Developer's Kit for Java Components

Variable	Notes

%CLASSPATH%	Includes:
	`.;%CLASSPATHJ%;%ORACLE_HOME%\lib\xmlparserv2.jar;`
	`%ORACLE_HOME%\lib\xsu12.jar;%ORACLE_HOME%\lib\xml.jar;`
	`%ORACLE_HOME%\lib\xmlmesg.jar;%ORACLE_HOME%\lib\oraclexsql.jar`
	Note: A single period "." to represent the current directory is not required, but may be useful.

%CLASSPATHJ%	For JDK 5, set: `CLASSPATHJ=%ORACLE_HOME%\jdbc\lib\ojdbc6.jar:%ORACLE_HOME%\lib\orai18n.jar` The `orai18n.jar` is needed to support certain character sets.
%JAVA_HOME%	Installation directory for the Java software developer's kit (SDK), Standard Edition. Modify the path that links to the Java SDK.
%PATH%	`%JAVA_HOME%\bin`

After setting up the XDK for Java environment on Windows, you can use the command-line utilities described in Table 11-5.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 11-5 Oracle XML Developer's Kit for Java Windows Utilities

Batch File/Class	Directory/JAR	Description
`xsql.bat`	`%ORACLE_HOME%\bin`	XSQL command-line utility. The batch file executes the `oracle.xml.xsql.XSQLCommandLine` class. Edit the batch file for your environment before use.
`OracleXML`	`%ORACLE_HOME%\lib\xsu12.jar`	XSU command-line utility
`orajaxb.bat`	`%ORACLE_HOME%\bin`	JAXB command-line utility
`orapipe.bat`	`%ORACLE_HOME%\bin`	Pipeline command-line utility
`oraxml.bat`	`%ORACLE_HOME%\bin`	XML parser command-line utility
`oraxsl.bat`	`%ORACLE_HOME%\bin`	XSLT processor command-line utility
`transx.bat`	`%ORACLE_HOME%\bin`	TransX command-line utility

Related Topics

- Using the XSQL Pages Command-Line Utility
  XDK includes a command-line Java interface that runs the XSQL page processor. You can process any XSQL page with the XSQL command-line utility.
- Using the XSU Command-Line Utility
  XDK includes a command-line Java interface for XSU. XSU command-line options are provided through the Java class `OracleXML`.
- Using the JAXB Class Generator Command-Line Utility
  XDK includes `orajaxb`, which is a command-line Java interface that generates Java classes from input XML schemas. Shell scripts `$ORACLE_HOME/bin/orajaxb` and `%ORACLE_HOME%\bin\orajaxb.bat` execute class `oracle.xml.jaxb.orajaxb`.
- Using the XML Pipeline Processor Command-Line Utility
  The command-line interface for the XML Pipeline processor is named `orapipe`. The Pipeline processor is packaged with Oracle Database. By default, the Oracle Universal Installer installs the utility on disk in `$ORACLE_HOME/bin`.
- Using the Java XML Parser Command-Line Utility (oraxml)
  The `oraxml` utility, which is located in `$ORACLE_HOME/bin` (UNIX) or `%ORACLE_HOME%\bin` (Windows), is a command-line interface that parses XML documents. It checks for both well-formedness and validity.
- Using the XSLT Processor Command-Line Utility
  XDK includes `oraxsl`, which is a command-line Java interface that can apply a stylesheet to multiple XML documents. The `$ORACLE_HOME/bin/oraxsl` and `%ORACLE_HOME%\bin\oraxsl.bat` shell scripts execute the `oracle.xml.jaxb.oraxsl` class.
- Using the TransX Command-Line Utility
  TransX utility is packaged with Oracle Database. By default, the Oracle Universal Installer installs the utility on disk.

11.3.4 Testing the XDK for Java Environment on Windows

An Microsoft Windows script is provided for testing the XDK for Java environment.

If your environment is set up correctly, then you can run the commands in Example 11-3 at the system prompt to generate version and usage information for the utilities in Table 11-5.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Example 11-3 Testing the Oracle XML Developer's Kit for Java Environment on Windows

```
xsql.bat
java OracleXML
orajaxb.bat -version
orapipe.bat -version
oraxsl.bat
transx.bat
```

11.4 Verifying the XDK (Java) Version

You can use `javac` to check your XDK version.

To see which version of XDK you have installed, use `javac` to compile the Java code shown in Example 11-4.

After compilation, run the program on the operating system command line:

```
java XDKVersion
```

The result is similar to:

```
You are using version:
Oracle XML Developers Kit 11.1.0.6.0 - Production
```

Example 11-4 XDKVersion.java

```
//
// XDKVersion.java
//
import java.net.URL;
import oracle.xml.parser.v2.XMLParser;
public class XDKVersion
{
   static public void main(String[] argv)
   {
      System.out.println("You are using version: ");
      System.out.println(XMLParser.getReleaseVersion());
   }
}
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

12

XML Parsing for Java

Extensible Markup Language (XML) parsing for Java is described.

12.1 Introduction to XML Parsing for Java

XML parsing for Java is described.

12.1.1 Prerequisites for Parsing with Java

An Oracle XML parser reads an XML document and uses either a Document Object Model (DOM) application programming interface (API) or Simple API for XML (SAX) to access to its content and structure. You can parse in either validating or nonvalidating mode.

This chapter assumes that you are familiar with these technologies:

- Document Object Model (DOM): An in-memory tree representation of the structure of an XML document.
- Simple API for XML (SAX): A standard for event-based XML parsing.
- Java API for XML Processing (JAXP): A standard interface for processing XML with Java applications that supports the DOM and SAX standards.
- document type definition (DTD): A set of rules that defines the valid structure of an XML document.
- XML Schema: A World Wide Web Consortium (W3C) recommendation that defines the valid structure of data types in an XML document.
- XML Namespaces: A mechanism for differentiating element and attribute names within an XML document.
- binary XML: An XML representation that uses the compact schema-aware format, in which both scalable and nonscalable DOMs can save XML documents.

For more information, see the list of XML resources in the Related Documents.

12.1.2 Standards and Specifications for XML Parsing for Java

The DOM Level 1, Level 2, and Level 3 specifications are W3C Recommendations.

See Document Object Model (DOM) Technical Reports for the W3C DOM specifications.

SAX is available in version 1.0 (deprecated) and 2.0. SAX is not a W3C specification. See SAX Project.

XML Namespaces are a W3C Recommendation. See Namespaces in XML 1.0 (Third Edition).

> **See Also**
>
> *Oracle XML DB Developer's Guide*

Copyright © 1999, 2025, Oracle and/or its affiliates.

JAXP is a standard API that enables the use of DOM, SAX, XML Schema, and Extensible Stylesheet Language Transformation (XSLT), independent of processor implementation.

> **See Also**
>
> Oracle XML Developer's Kit Standards, for information about standards supported by Oracle XML Developer's Kit (XDK)

12.1.3 Large Node Handling

DOM Stream access to XML nodes is done by Procedural Language/Structured Query Language (PL/SQL) and Java APIs. Nodes in an XML document can now far exceed 64 KB. Thus Joint Photographic Experts Group (JPEG), Word, PDF, rich text format (RTF), and HTML documents can be more readily stored.

> **See Also**
>
> *Oracle XML DB Developer's Guide* for complete details on the Java large node capabilities

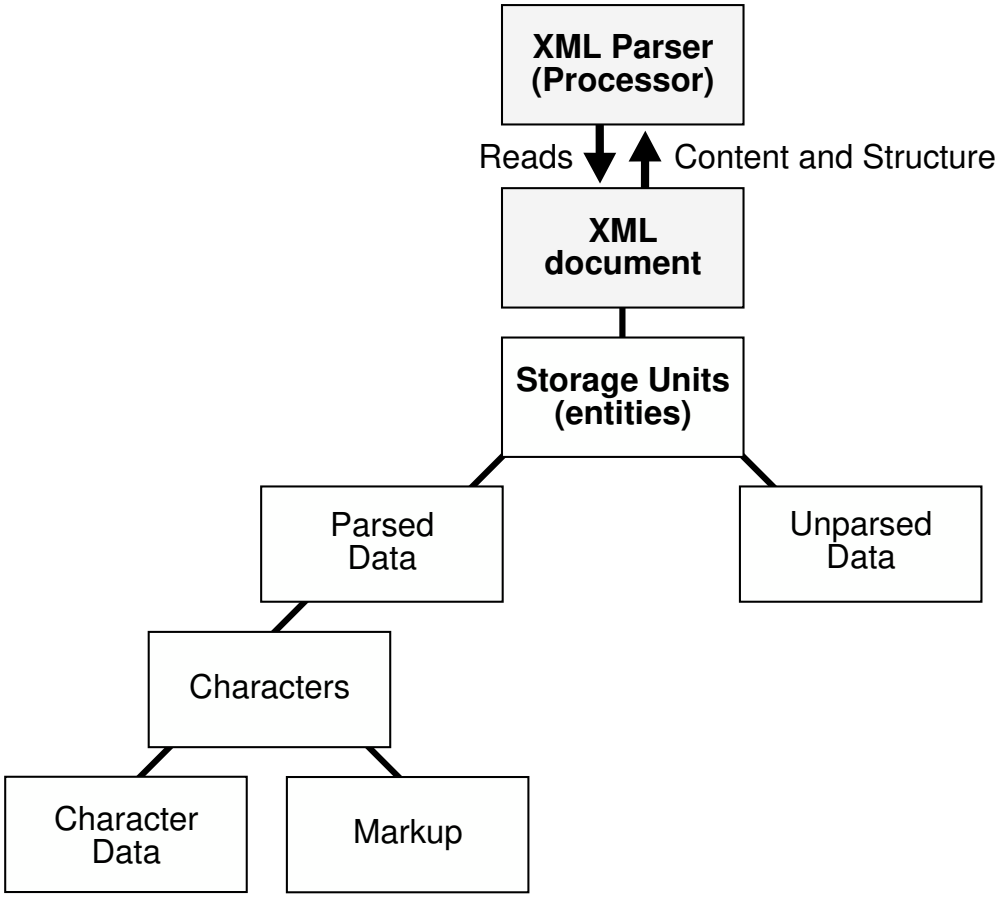
12.1.4 XML Parsing in Java: Overview

`XMLParser` is the abstract base class for the XML parser for Java. An instantiated parser invokes the `parse()` method to read an XML document. `XMLDOMImplementation` factory methods provide another way to parse binary XML to create scalable DOM.

Figure 12-1 shows the basic parsing process, using `XMLParser`. The figure does not apply to `XMLDOMImplementation()`.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Figure 12-1 XML Parser Process



These APIs provide a Java application with access to a parsed XML document:

- DOM API

  DOM API parses XML documents and builds a tree representation of the documents in memory. To parse with DOM API, use either a `DOMParser` object or the `XMLDOMImplementation` interface factory methods to create a pluggable, scalable DOM (SDOM).

- SAX API

  SAX API processes an XML document as a stream of events, which means that a program cannot access random locations in a document. To parse with SAX API, use a `SAXParser` object.

- JAXP

  JAXP is a Java-specific API that supports DOM, SAX, and Extensible Stylesheet Language (XSL). To parse with JAXP, use a `DocumentBuilder` or `SAXParser` object.

Subsequent topics use the sample XML document in Example 12-1 to show the differences among DOM, SAX, and JAXP.

Example 12-1 Sample XML Document

```
<?xml version="1.0"?>
  <EMPLIST>
    <EMP>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
  <ENAME>MARY</ENAME>
 </EMP>
 <EMP>
  <ENAME>SCOTT</ENAME>
 </EMP>
</EMPLIST>
```

12.1.5 DOM in XML Parsing

DOM API builds an in-memory tree representation of the XML document. DOM API provides classes and methods to navigate and process the tree.

For example, given the document described in Example 12-1, the DOM API creates the in-memory tree shown in Figure 12-2.

The important aspects of DOM API are:

- DOM API provides a familiar tree structure of objects, making it easier to use than the SAX API.
- The tree can be manipulated. For example, elements can be reordered and renamed, and both elements and attributes can be added and deleted.
- Interactive applications can store the tree in memory, where users can access and manipulate it.
- XKD includes DOM API extensions that support XPath. (Although the DOM standard does not support XPath, most XPath implementations use DOM.)
- XDK supports SDOM. For details, see SDOM.

12.1.5.1 DOM Creation

In XDK for Java, there are three ways to create a DOM: parse a document using `DOMParser`, use an `XMLDOMImplementation` factory method (creates a scalable DOM), or use an `XMLDocument` constructor (uncommon).

12.1.6 SDOM

XDK supports pluggable, scalable DOM (SDOM). This support relieves problems of memory inefficiency, limited scalability, and lack of control over the DOM configuration. SDOM creation and configuration are mainly supported using the `XMLDOMImplementation` class.

Important aspects of SDOM are:

- SDOM can use plug-in external XML in its existing forms.

  Plug-in XML data can be in different forms—binary XML, `XMLType`, third-party DOM, and so on. SDOM need not replicate external XML in an internal representation. SDOM is created on top of plug-in XML data through the `Reader` and `InfosetWriter` abstract interfaces.
- SDOM has transient nodes.

  Nodes are created only if they are accessed and are freed if they are not used.
- SDOM can use binary XML as both input and output.

  SDOM can interact with data in two ways:

  – Through the abstract `InfosetReader` and `InfosetWriter` interfaces.

Copyright © 1999, 2025, Oracle and/or its affiliates.

To read and write `BinXML` data, users can use the `BinXML` implementation of `InfosetReader` and `InfosetWriter`. To read and write in other forms of XML infoset, users can use their own implementations.

– Through an implementation of the `InfosetReader` and `InfosetWriter` adaptor for `BinXMLStream`.

Related Topics

- Using Binary XML with Java
  Topics here explain how to use Binary XML with Java.

12.1.6.1 Pluggable DOM Support

Pluggable DOM lets you split the DOM API from the data layer. The DOM API is separated from the data by the `InfosetReader` and `InfosetWriter` interfaces. Using pluggable DOM, you can easily move XML data from one processor to another.

The DOM API includes unified standard APIs on top of the data to support node access, navigation, update processes, and searching capability.

Related Topics

- Using SDOM
  How to use SDOM is described.

12.1.6.2 Lazy Materialization

Using lazy materialization, XDK creates only nodes that are accessed and frees unused nodes from memory. Applications can process very large XML documents with improved scalability.

Related Topics

- Using Lazy Materialization
  Using lazy materialization, you can plug in an empty DOM, which can pull in data when needed and free (dereference) nodes when they are no longer needed. SDOM supports either manual or automatic node dereferencing.

12.1.6.3 Configurable DOM Settings

DOM configurations can be made to suit different applications. You can configure the DOM with different access patterns such as read-only, streaming, transient update, and shadow copy, achieving maximum memory use and performance in your applications.

Related Topics

- Using Configurable DOM Settings
  When you create a DOM using class `XMLDOMImplementation`, you can configure the DOM for different applications and achieve maximum efficiency by using method `setAttribute`.

12.1.6.4 DOM Support for Fast Infoset

Fast Infoset, developed by Oracle, is a compact binary XML format that represents the XML Infoset. This format has become the international standard ITU-T SG 17 and ISO/IEC JTC1 SC6. The Fast Infoset representation of XML Infoset is popular within the Java XML and Web Service communities.

Fast Infoset provides these benefits in comparison with other formats:

- It is more compact, parses faster, and serializes better than XML text.

Copyright © 1999, 2025, Oracle and/or its affiliates.

- It encodes and decodes faster than parsing of XML text, and Fast Infoset documents are generally 20 to 60 percent smaller than the corresponding XML text.
- It leads other binary XML formats in performance and compression ratio, and handles small to large documents in a more balanced manner.

SDOM is the XDK DOM configuration that supports scalability. It is built on top of serialized binary data to provide a DOM API to applications like XPath and XSLT. SDOM has an open plug-in architecture that reads binary data through an abstract API InfosetReader. The InfosetReader API allows SDOM to decode the binary data going forward, remember the start location of the nodes, and search a location to decode from there. This support enables SDOM to free nodes that are not in use and re-create those nodes from binary data when they are needed. When binary data is stored externally, such as in a file or a BLOB, SDOM is highly scalable.

Related Topics

- Using Fast Infoset with SDOM
  The Fast Infoset to XDK/J model lets you use Fast Infoset techniques while working with XML content in Java.

12.1.7 SAX in the XML Parser

Unlike DOM, SAX is event-based, so SAX API does not build in-memory tree representations of input documents. SAX API processes the input document element by element and can report events and significant data to callback methods in the application.

For example, given the document described in Example 12-1, the SAX API parses it as the series of linear events shown in Figure 12-2.

The important aspects of SAX API are:

- It is useful for search operations and other programs that need not manipulate an XML tree.
- It does not consume significant memory resources.
- It is faster than DOM when retrieving XML documents from a database.

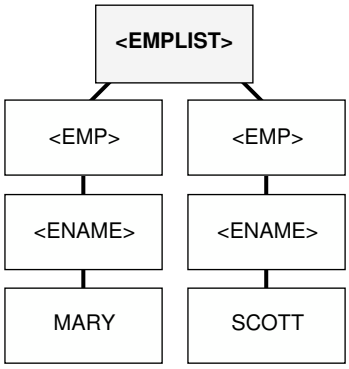
Copyright © 1999, 2025, Oracle and/or its affiliates.

Figure 12-2 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: ENAME
end element: EMP

start element: EMP
start element: ENAME
characters: SCOTT
end element: ENAME
end element: EMP

end element: EMPLIST
end document
```

Useful for applications such as search and retrieval that do not change the "XML tree".

12.1.8 JAXP in the XML Parser

JAXP lets you plug in an implementation of the SAX or DOM parser. The SAX and DOM APIs provided by XDK are examples of vendor-specific implementations supported by JAXP. The main advantage of JAXP is that it lets you write interoperable applications.

An application that uses features available through JAXP can very easily switch the implementation.

The main disadvantage of JAXP is that it runs more slowly than vendor-specific APIs. Also, JAXP lacks several features that Oracle-specific APIs provide. Some Oracle-specific features are available through the JAXP extension mechanism, but an application that uses these extensions loses the flexibility of switching implementation.

12.1.9 Namespace Support in the XML Parser

Namespaces can help you avoid name collisions between elements or attributes in XML documents.

Example 12-2 is an XML document that uses the `<address>` tag for both a company address and an employee address. An XML processor cannot distinguish between a company address and an employee address.

Example 12-3 is an XML document that uses these namespaces to distinguish between company and employee `<address>` tags:

```
http://www.oracle.com/employee
http://www.oracle.com/company
```

Example 12-3 associates the `com` prefix with the first namespace and the `emp` prefix with the second namespace.

When parsing documents that use namespaces, it is helpful to remember these terms:

Copyright © 1999, 2025, Oracle and/or its affiliates.

- **Namespace URI** is the URI assigned to `xmlns`. In Example 12-3, `http://www.oracle.com/employee` and `http://www.oracle.com/company` are namespace URIs.
- **Namespace prefix** is a namespace identifier declared with `xmlns`. In Example 12-3, `emp` and `com` are namespace prefixes.
- **Local name** is the name of an element or attribute without the namespace prefix. In Example 12-3, `employee` and `company` are local names.
- **Qualified name** is the local name plus the prefix. In Example 12-3, `emp:employee` and `com:company` are qualified names.
- **Expanded name** is the result of substituting the namespace URI for the namespace prefix. In Example 12-3, `http://www.oracle.com/employee:employee` and `http://www.oracle.com/company:company` are expanded element names.

Example 12-2 Sample XML Document Without Namespaces

```
<?xml version='1.0'?>
<addresslist>
  <company>
    <address>500 Oracle Parkway,
             Redwood Shores, CA 94065
    </address>
  </company>
  <!-- ... -->
  <employee>
    <lastname>King</lastname>
    <address>3290 W Big Beaver
             Troy, MI 48084
    </address>
  </employee>
  <!-- ... -->
</addresslist>
```

Example 12-3 Sample XML Document with Namespaces

```
<?xml version='1.0'?>
<addresslist>
<!-- ... -->
  <com:company
    xmlns:com="http://www.oracle.com/company">
    <com:address>500 Oracle Parkway,
             Redwood Shores, CA 94065
    </com:address>
  </com:company>
  <!-- ... -->
  <emp:employee
    xmlns:emp="http://www.oracle.com/employee">
    <emp:lastname>King</emp:lastname>
    <emp:address>3290 W Big Beaver
             Troy, MI 48084
    </emp:address>
</emp:employee>
```

12.1.10 Validation in the XML Parser

To parse an XML document, invoke the `parse()` method. Typically, you invoke initialization and termination methods in association with the `parse()` method.

The parser mode can be either validating or nonvalidating. In validating mode, the parser determines whether the document conforms to the specified DTD or XML schema. In

Copyright © 1999, 2025, Oracle and/or its affiliates.

nonvalidating mode, the parser checks only for well-formedness. To set the parser mode, invoke the `setValidationMode()` method defined in `oracle.xml.parser.v2.XMLParser`.

Table 12-1 shows the `setValidationMode()` flags that you can use in the XDK parser.

Table 12-1 XML Parser for Java Validation Modes

Name	Value	The XML Parser . . .
Nonvalidating mode	`NONVALIDATING`	Verifies that the XML is well-formed and parses the data.
DTD validating mode	`DTD_VALIDATION`	Verifies that the XML is well-formed and validates the XML data against the DTD. The DTD defined in the `<!DOCTYPE>` declaration must be relative to the location of the input XML document.
Schema validation mode	`SCHEMA_VALIDATION`	Validates the XML Document according to the XML schema specified for the document.
LAX validation mode	`SCHEMA_LAX_VALIDATION`	Tries to validate part or all of the instance document if it can find the schema definition. It does not raise an error if it cannot find the definition. See the sample program `XSDLax.java` in the `schema` directory.
Strict validation mode	`SCHEMA_STRICT_VALIDATION`	Tries to validate the whole instance document, raising errors if it cannot find the schema definition or if the instance does not conform to the definition.
Partial validation mode	`PARTIAL_VALIDATION`	Validates all or part of the input XML document according to the DTD, if present. If the DTD is not present, then the parser is set to nonvalidating mode.
Auto validation mode	`AUTO_VALIDATION`	Validates all or part of the input XML document according to the DTD or XML schema, if present. If neither is present, then the parser is set to nonvalidating mode.

In addition to setting the validation mode with `setValidationMode()`, you can use the `oracle.xml.parser.schema.XSDBuilder` class to build an XML schema and then configure the parser to use it by invoking the `XMLParser.setXMLSchema()` method. In this case, the XML parser automatically sets the validation mode to `SCHEMA_STRICT_VALIDATION` and ignores the `schemaLocation` and `noNamespaceSchemaLocation` attributes. You can also change the validation mode to `SCHEMA_LAX_VALIDATION`. The `XMLParser.setDoctype()` method is a parallel method for DTDs, but unlike `setXMLSchema()` it does not alter the validation mode.

See Also

- Using the XML Schema Processor for Java to learn about validation
- *Oracle AI Database XML Java API Reference* to learn about the `XMLParser` and `XSDBuilder` classes

12.1.11 Compression in the XML Parser

You can use the XML compressor, which is implemented in the XML parser, to compress and decompress XML documents. The compression algorithm is based on tokenizing the XML tags.

Copyright © 1999, 2025, Oracle and/or its affiliates.

The assumption is that any XML document repeats some tags, so tokenizing these tags gives considerable compression. The degree of compression depends on the type of document: the larger the tags and the lesser the text content, the better the compression.

The Oracle XML parser generates a binary compressed output from either an in-memory DOM tree or SAX events generated from an XML document. Table 12-2 describes the two types of compression.

Table 12-2 XML Compression with DOM and SAX

Type	Description	Compression APIs
DOM-based	The goal is to reduce the size of the XML document without losing the structural and hierarchical information of the DOM tree. The parser serializes an in-memory DOM tree, corresponding to a parsed XML document, and generates a compressed XML output stream. The serialized stream regenerates the DOM tree when read back.	Use the `writeExternal()` method to generate compressed XML and the `readExternal()` method to reconstruct it. The methods are in the `oracle.xml.parser.v2.XMLDocument` class.
SAX-based	The SAX parser generates a compressed stream when it parses an XML file. SAX events generated by the SAX parser are handled by the SAX compression utility, which generates a compressed binary stream. When the binary stream is read back, the SAX events are generated.	To generate compressed XML, instantiate `oracle.xml.comp.CXMLHandlerBase` by passing an output stream to the constructor. Pass the object to `SAXParser.setContentHandler()` and then execute the `parse()` method. Use the `oracle.xml.comp.CXMLParser` class to decompress the XML. **Note:** `CXMLHandlerBase` implements both SAX 1.0 and 2.0, but because 1.0 is deprecated, Oracle recommends that you use the 2.0 API.

The compressed streams generated from DOM and SAX are compatible; that is, you can use the compressed stream generated from SAX to generate the DOM tree and the reverse. As with XML documents in general, you can store the compressed XML data output in the database as a `BLOB` data item.

When a program parses a large XML document and creates a DOM tree in memory, it can affect performance. You can compress an XML document into a binary stream by serializing the DOM tree. You can regenerate the DOM tree without validating the XML data in the compressed stream. You can treat the compressed stream as a serialized stream, but the data in the stream is more controlled and managed than the compression implemented by Java default serialization.

> **Note**
>
> Oracle Text cannot search a compressed XML document. Decompression reduces performance. If you are transferring files between client and server, then Hypertext Transfer Protocol (HTTP) compression can be easier.

Copyright © 1999, 2025, Oracle and/or its affiliates.

12.2 Using XML Parsing for Java: Overview

The fundamental component of any XML development is XML parsing. XML parsing for Java is a standalone XML component that parses an XML document (and possibly also a standalone DTD or XML schema) so that your program can process it.

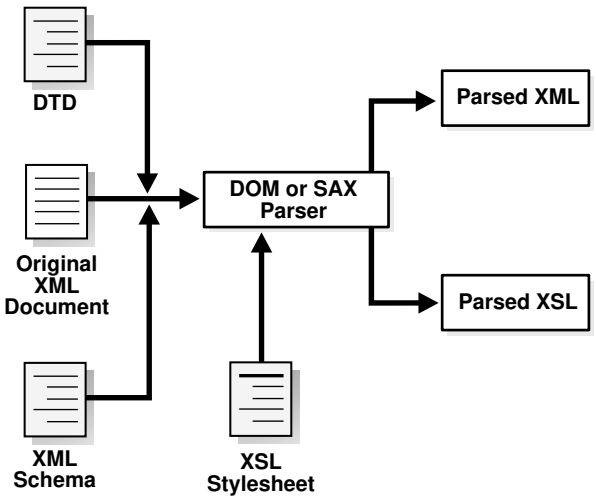
> **Note**
>
> You can use the parser with any supported Java Virtual Machine (JVM). With Oracle 9i or later, you can load the parser into the database and use the internal Oracle JVM. For other database versions, run the parser in an external JVM and connect to a database through JDBC.

12.2.1 Using the XML Parser for Java: Basic Process

Thd basic process of using the XML Parser for Java is described.

Figure 12-3 shows how to use the XML parser in a typical XML processing application.

Figure 12-3 XML Parser for Java



The basic process of the application shown in Figure 12-3 is:

1. The DOM or SAX parser parses input XML documents. For example, the program can parse XML data documents, DTDs, XML schemas, and XSL stylesheets.
2. If you implement a validating parser, then the processor attempts to validate the XML data document against any supplied DTDs or XML schemas.

> **See Also**
>
> *Oracle AI Database XML Java API Reference* for XML parser classes and methods

Copyright © 1999, 2025, Oracle and/or its affiliates.

12.2.2 Running the XML Parser for Java Demo Programs

Demo programs for the XML parser for Java are included in `$ORACLE_HOME/xdk/demo/java/parser`.

The demo programs are distributed among the subdirectories described in Table 12-3.

Table 12-3 Java Parser Demos

Directory	Contents	These programs ...

`common`	`class.xml`
	`DemoUtil.java`
	`empl.xml`
	`family.dtd`
	`family.xml`
	`iden.xsl`
	`NSExample.xml`
	`traversal.xml`
		Provide XML files and Java programs for general use with the XML parser. For example, you can use the XSLT stylesheet `iden.xsl` to achieve an identity transformation of the XML files. `DemoUtil.java` implements a helper method to create a URL from a file name, and is used by many other demo programs.

`comp`	`DOMCompression.java`
	`DOMDeCompression.java`
	`SAXCompression.java`
	`SAXDeCompression.java`
	`SampleSAXHandler.java`
	`sample.xml`
	`xml.ser`
		Show DOM and SAX compression:
		• `DOMCompression.java` compresses a DOM tree.
		• `DOMDeCompression.java` reads back a DOM from a compressed stream.
		• `SAXCompression.java` compresses the output from a SAX parser.
		• `SAXDeCompression.java` regenerates SAX events from the compressed stream.
		• `SampleSAXHandler.java` shows use of a handler to handle the events thrown by the SAX `DeCompressor`.

`dom`	`AutoDetectEncoding.java`
	`DOM2Namespace.java`
	`DOMNamespace.java`
	`DOMRangeSample.java`
	`DOMSample.java`
	`EventSample.java`
	`I18nSafeXMLFileWritingSample.java`
	`NodeIteratorSample.java`
	`ParseXMLFromString.java`
	`TreeWalkerSample.java`
		Show uses of the DOM API:
		• `DOM2Namespace.java` shows how to use DOM Level 2.0 APIs.
		• `DOMNamespace.java` shows how to use Namespace extensions to DOM APIs.
		• `DOMRangeSample.java` shows how to use DOM Range APIs.
		• `DOMSample.java` shows basic use of the DOM APIs.
		• `EventSample.java` shows how to use DOM Event APIs.
		• `NodeIteratorSample.java` shows how to use DOM Iterator APIs.
		• `TreeWalkerSample.java` shows how to use DOM `TreeWalker` APIs.

`jaxp`	`JAXPExamples.java`
	`age.xsl`
	`general.xml`
	`jaxpone.xml`
	`jaxpone.xsl`
	`jaxpthree.xsl`
	`jaxptwo.xsl`
	`oraContentHandler.java`
		Show various uses of the JAXP:
		• `JAXPExamples.java` provides a few examples of how to use the JAXP 1.1 API to run the Oracle engine.
		• `oraContentHandler.java` implements a SAX content handler. The program invokes methods such as `startDocument()`, `endDocument()`, `startElement()`, and `endElement()` when it recognizes an XML tag.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 12-3 (Cont.) Java Parser Demos

Directory	Contents	These programs ...
sax	SAX2Namespace.java SAXNamespace.java SAXSample.java Tokenizer.java	Show various uses of the SAX APIs: • SAX2Namespace.java shows how to use SAX 2.0. • SAXNamespace.java shows how to use namespace extensions to SAX APIs. • SAXSample.java shows basic use of the SAX APIs. • Tokenizer.java shows how to use the XMLToken interface APIs. The program implements the XMLToken interface, which must be registered with the setTokenHandler() method. A request for XML tokens is registered with the setToken() method. During tokenizing, the parser does not validate the document and does not include or read internal/external utilities.
xslt	XSLSample.java XSLSample2.java match.xml match.xsl math.xml math.xsl number.xml number.xsl position.xml position.xsl reverse.xml reverse.xsl string.xml string.xsl style.txt variable.xml variable.xsl	Show the transformation of documents with XSLT: • XSLSample.java shows how to use the XSL processing capabilities of the Oracle XML parser. It transforms an input XML document with a given input stylesheet. This demo builds the result of XSL transformations as a DocumentFragment and so does not support xsl:output features. • XSLSample2.java transforms an input XML document with a given input stylesheet. The demo streams the result of the XSL transformation and so supports xsl:output features. **See Also:** Running the XSLT Processor Demo Programs

Documentation for how to compile and run the sample programs is located in the README file. The basic procedure is:

1. Change into the $ORACLE_HOME/xdk/demo/java/parser directory (UNIX) or %ORACLE_HOME%\xdk\demo\java\parser directory (Windows).
2. Set up your environment as described in Setting Up the XDK for Java Environment.
3. Change into each of these subdirectories and run make (UNIX) or Make.bat (Windows) at the command line. For example:

   ```
   cd comp;make;cd ..
   cd jaxp;make;cd ..
   cd sax;make;cd ..
   cd dom;make;cd ..
   cd xslt;make;cd ..
   ```

   The make file compiles the source code in each directory, runs the programs, and writes the output for each program to a file with an *.out extension.
4. You can view the *.out files to view the output for the programs.

Copyright © 1999, 2025, Oracle and/or its affiliates.

12.2.3 Using the Java XML Parser Command-Line Utility (oraxml)

The `oraxml` utility, which is located in `$ORACLE_HOME/bin` (UNIX) or `%ORACLE_HOME%\bin` (Windows), is a command-line interface that parses XML documents. It checks for both well-formedness and validity.

To use `oraxml`, ensure that:

- Your `CLASSPATH` is set up as described in Setting Up the XDK for Java Environment, and your `CLASSPATH` environment variable references the `xmlparserv2.jar` file.
- Your `PATH` environment variable can find the Java interpreter that comes with your version of the Java Development Kit (JDK).

Table 12-4 lists the `oraxml` command-line options.

Table 12-4 oraxml Command-Line Options

Option	Purpose
`-help`	Prints the help message
`-version`	Prints the release version
`-novalidate` *`fileName`*	Checks whether the input file is well-formed
`-dtd` *`fileName`*	Validates the input file with DTD Validation
`-schema` *`fileName`*	Validates the input file with Schema Validation
`-log` *`logfile`*	Writes the errors to the output log file
`-comp` *`fileName`*	Compresses the input XML file
`-decomp` *`fileName`*	Decompresses the input compressed file
`-enc` *`fileName`*	Prints the encoding of the input file
`-warning`	Shows warnings

For example, change into the `$ORACLE_HOME/xdk/demo/java/parser/common` directory. You can validate the document `family.xml` against `family.dtd` by executing this command on the command line:

```
oraxml -dtd -enc family.xml
```

The output is:

```
The encoding of the input file: UTF-8

The input XML file is parsed without errors using DTD validation mode.
```

12.3 Parsing XML with DOM

The W3C standard library `org.w3c.dom` defines the `Document` class and classes for the components of a DOM. The Oracle XML parser includes the standard DOM APIs and complies with the W3C DOM recommendation.

Along with `org.w3c.dom`, Oracle XML parsing includes classes that implement the DOM APIs and extends them to provide features such as printing document fragments and retrieving namespace information.

Copyright © 1999, 2025, Oracle and/or its affiliates.

12.3.1 Using the DOM API for Java

Java classes that you can use to implement DOM-based components in your XML application are described.

Use these classes:

- `oracle.xml.parser.v2.DOMParser`

  This class implements an XML 1.0 parser according to the W3C recommendation. Because `DOMParser` extends `XMLParser`, all methods of `XMLParser` are available to `DOMParser`.

- `oracle.xml.parser.v2.XMLDOMImplementation`

  This class contains factory methods used to create SDOM.

You can also use the `DOMNamespace` and `DOM2Namespace` classes, which are sample programs included in `$ORACLE_HOME/xdk/demo/java/parser/dom`.

Related Topics

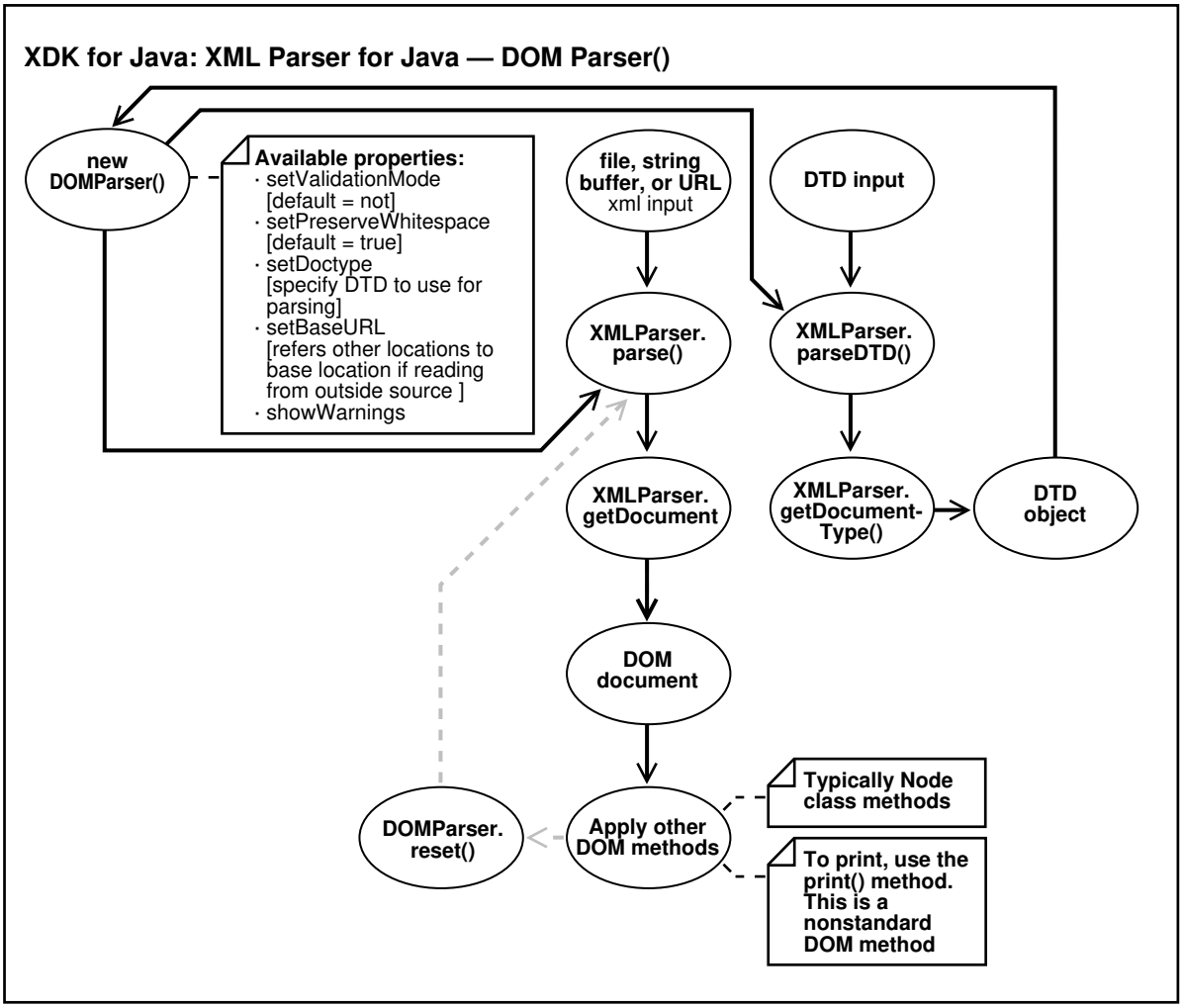
- DOM Parser Architecture
  The architecture of the DOM Parser is described.
- Creating SDOM
  How to create and use a pluggable, scalable DOM (SDOM) is explained.

12.3.2 DOM Parser Architecture

The architecture of the DOM Parser is described.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Figure 12-4 Basic Architecture of the DOM Parser



12.3.3 Performing Basic DOM Parsing

`DOMSample.java` shows the basic steps for parsing an input XML document and accessing it through a DOM. `DOMSample.java` receives an XML file as input, parses it, and prints the elements and attributes in the DOM tree.

The steps, which provide possible methods and interfaces that you can use, are:

1. Create a `DOMParser` object (a parser) by invoking the `DOMParser()` constructor.

   The code in `DOMSample.java` is:

   ```
   DOMParser parser = new DOMParser();
   ```

2. Configure the parser properties, using the methods in Table 12-5.

   This code fragment from `DOMSample.java` specifies the error output stream, sets the validation mode to DTD validation, and enables warning messages:

   ```
   parser.setErrorStream(System.err);
   parser.setValidationMode(DOMParser.DTD_VALIDATION);
   parser.showWarnings(true);
   ```


Copyright © 1999, 2025, Oracle and/or its affiliates.

3. Parse the input XML document by invoking the `parse()` method, which builds a tree of `Node` objects in memory.

   This code fragment from `DOMSample.java` parses an instance of the `java.net.URL` class:

   ```
   parser.parse(url);
   ```

   The XML input can be a file, a string buffer, or a URL. As the following code fragment shows, `DOMSample.java` accepts a file name as a parameter and invokes the `createURL` helper method to construct a URL object that can be passed to the parser:

   ```
   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);
            }
            ...
            // Generate a URL from the filename.
            URL url = DemoUtil.createURL(argv[0]);
            ...
   ```

4. Invoke `getDocument()` to get a handle to the root of the in-memory DOM tree, which is an `XMLDocument` object.

   You can use this handle to access every part of the parsed XML document. The `XMLDocument` class implements the interfaces in Table 12-6.

   The code fragment from `DOMSample.java` is:

   ```
   XMLDocument doc = parser.getDocument();
   ```

5. Get and manipulate DOM nodes of the retrieved document by invoking `XMLDocument` methods in Table 12-7.

   This code fragment from `DOMSample.java` uses the `DOMParser.print()` method to print the elements and attributes of the DOM tree:

   ```
   System.out.print("The elements are: ");
   printElements(doc);

   System.out.println("The attributes of each element are: ");
   printElementAttributes(doc);
   ```

   The following code fragment from `DOMSample.java` implements the `printElements()` method, which invokes `getElementsByTagName()` to get a list of all the elements in the DOM tree. Then the code loops through the list, invoking `getNodeName()` to print the name of each element:

   ```
   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() + " ");
   ```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
   }

   System.out.println();
}
```

The following code fragment from `DOMSample.java` implements the `printElementAttributes()` method, which invokes `Document.getElementsByTagName()` to get a list of all the elements in the DOM tree. Then the code loops through the list, invoking `Element.getAttributes()` to get the list of attributes for the element and invoking `Node.getNodeName()` to get the attribute name and `Node.getNodeValue()` to get the attribute value:

```
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<nnm.getLength(); i++)
         {
            n = nnm.item(i);
            attrname = n.getNodeName();
            attrval = n.getNodeValue();
            System.out.print(" " + attrname + " = " + attrval);
         }
      }
      System.out.println();
   }
}
```

6. Reset the parser state by invoking the `reset()` method. The parser is now ready to parse a new document.

Table 12-5 summarizes the `DOMParser` configuration methods.

Table 12-5 DOMParser Configuration Methods

Method	Purpose
`setBaseURL()`	Sets the base URL for loading external entities and DTDs. Invoke this method if the XML document is an `InputStream`.
`setDoctype()`	Specifies the DTD to use when parsing.
`setErrorStream()`	Creates an output stream for the output of errors and warnings.
`setPreserveWhitespace()`	Instructs the parser to preserve the white space in the input XML document.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 12-5 (Cont.) DOMParser Configuration Methods

Method	Purpose
`setValidationMode()`	Sets the validation mode of the parser. Table 12-1 describes the flags that you can use with this method.
`showWarnings()`	Specifies whether the parser prints warnings.

Table 12-6 summarizes the interfaces that the `XMLDocument` class implements.

Table 12-6 Some Interfaces Implemented by XMLDocument

Interface	What Interface Defines
`org.w3c.dom.Node`	A single node in the document tree and methods to access and process the node.
`org.w3c.dom.Document`	A `Node` that represents the entire XML document.
`org.w3c.dom.Element`	A `Node` that represents an XML element.

Table 12-7 summarizes the methods for getting and manipulating DOM tree nodes.

Table 12-7 Methods for Getting and Manipulating DOM Tree Nodes

Method	Purpose
`getAttributes()`	Generates a `NamedNodeMap` containing the attributes of this node (if it is an element) or `null` otherwise.
`getElementsbyTagName()`	Retrieves recursively all elements that match a given tag name under a certain level. This method supports the * tag, which matches any tag. Invoke `getElementsByTagName("*")` through the handle to the root of the document to generate a list of all elements in the document.
`getExpandedName()`	Gets the expanded name of the element. This method is specified in the `NSName` interface.
`getLocalName()`	Gets the local name for this element. If an element name is `<E1:locn xmlns:E1="http://www.oracle.com/"/>`, then `locn` is the local name.
`getNamespaceURI()`	Gets the namespace URI of this node, or `null` if it is unspecified. If an element name is `<E1:locn xmlns:E1="http://www.oracle.com/"/>`, then `http://www.oracle.com` is the namespace URI.
`getNodeName()`	Gets the name of a node in the DOM tree.
`getNodeValue()`	Gets the value of this node, depending on its type. This node is in the `Node` interface.
`getPrefix()`	Gets the namespace prefix for an element.
`getQualifiedName()`	Gets the qualified name for an element. If an element name is `<E1:locn xmlns:E1="http://www.oracle.com/"/>`, then `E1:locn` is the qualified name..
`getTagName()`	Gets the name of an element in the DOM tree.

Copyright © 1999, 2025, Oracle and/or its affiliates.

12.3.4 Creating SDOM

How to create and use a pluggable, scalable DOM (SDOM) is explained.

12.3.4.1 Using SDOM

How to use SDOM is described.

SDOM has the DOM API split from the data. The underlying data can be either internal data or plug-in data, and both can be in binary XML.

Internal data is XML text that has not been parsed. To be plugged in, internal data must be saved as binary XML and then parsed by the `DOMParser`. The parsed binary XML can be then be plugged into the `InfoSetReader` of the DOM API layer. The `InfosetReader` argument is the interface to the underlying XML data.

Plug-in data is XML text that has been parsed, and can therefore be transferred from one processor to another.

To create an SDOM, you plug in XML data through the `InfosetReader` API on an `XMLDOMImplementation` object. For example:

```
public Document createDocument(InfosetReader reader) throws DOMException
```

The `InfosetReader` API is implemented on top of `BinXMLStream`. Optional adaptors for other forms of XML data (such as dom4j, JDOM, or Java Database Connectivity (JDBC)) may also be supported. You can also plug in your own implementations.

`InfosetReader` serves as the interface between the scalable DOM API layer and the underlying data. It is a generic, stream-based pull API that accesses XML data. The `InfosetReader` retrieves sequential events from the XML stream and queries the state and data from these events. The following code scans the XML data and retrieves the `QNames` and attributes of all elements:

```
InfosetReader reader;
While (reader.hasNext())
{
   reader.next();
   if (reader.getEventType() == START_ELEMENT)
   {
        QName name = reader.getQName();
        TypedAttributeList attrList = reader.getAttributeList();
     }
}
```

12.3.4.1.1 InfosetReader Options

Options supported by the `InfosetReader` API are presented.

These are the supported operations:

- Copying (Optional, but `InfosetReader` from `BinXMLStream` always supports it)

Copyright © 1999, 2025, Oracle and/or its affiliates.

To support shadow copying of DOM across documents, you can create a new copy of `InfosetReader` to ensure thread safety, using the `Clone` method. For more information, see Using Shadow Copy.

- Moving Focus (Optional)

  To support lazy materialization, the `InfosetReader` may have the ability to move focus to any location specified by `offset`:

  ```
  If (reader.hasSeekSupport())
     reader.seek(offset);
  ```

  For more information, see Using Lazy Materialization

12.3.4.1.2 InfosetWriter

`InfosetWriter` is an extension of the `InfosetReader` API that supports data writing. XDK implements `InfosetWriter` on top of binary XML. You cannot modify this implementation.

12.3.4.1.3 Saving XML Text as Binary XML

To create a scalable DOM from XML text, you must save the XML text as either binary XML or references to binary XML before you can run `DOMParser` on it. To save the XML text as binary XML, set the `doc.save` argument to `false`.

```
XMLDocument doc;
InfosetWriter writer;
doc.save(writer, false);
writer.close();
```

If you know that the data source is available for deserialization, then you can save the section reference of binary XML instead of the actual data by setting the `doc.save` argument to `true`.

Related Topics

- Using Binary XML with Java
  Topics here explain how to use Binary XML with Java.

12.3.4.2 Using Lazy Materialization

Using lazy materialization, you can plug in an empty DOM, which can pull in data when needed and free (dereference) nodes when they are no longer needed. SDOM supports either manual or automatic node dereferencing.

12.3.4.2.1 Pulling Data on Demand

The plug-in DOM architecture creates an empty DOM, which contains a single `Document` node as the root of the tree. The rest of the DOM tree can be expanded later if it is accessed.

A node can have unexpanded child and sibling nodes, but its parent and ancestors are always expanded. Each node maintains the `InfoSetReader.Offset` property of the next node so that the DOM can pull additional data to create the next node.

Depending on access method type, DOM nodes can expand more than the set of nodes returned:

Copyright © 1999, 2025, Oracle and/or its affiliates.

Access Method	Description
DOM Navigation	Allows access to neighboring nodes such as first child, last child, parent, previous sibling, or next sibling. If node creation is needed, it is done in document order.
Identifier (ID) Indexing	A DTD or XML schema can specify nodes with the type ID. If the DOM supports ID indexing, those nodes can be directly retrieved using the index. In scalable DOM, retrieval by index does not cause the expansion of all previous nodes, but their ancestor nodes are materialized.
XPath Expressions	XPath evaluation can cause materialization of all intermediate nodes in memory. For example, the descendent axis '//' expands the whole subtree, although some nodes might be released after evaluation.

12.3.4.2.2 Using Automatic Node Dereferencing

DOM navigation support requires additional links between nodes. In automatic dereferencing mode, weak links can be automatically dereferenced during garbage collection. To use automatic node dereferencing, set the `PARTIAL_DOM` attribute to `Boolean.TRUE`.

Node release depends on link importance. Links to parent nodes cannot be dropped, because ancestors provide context for in-scope namespaces and it is difficult to retrieve dropped parent nodes using streaming APIs such as `InfosetReader`.

In an SDOM tree, links to parent and previous sibling nodes are strong and links to child and following sibling nodes are weak. When the JVM frees the nodes, references to them are still available in the underlying data so they can be re-created if needed.

12.3.4.2.3 Using Manual Node Dereferencing

Manual node dereferencing is described.

In manual dereferencing mode, there are no weak references. The application must explicitly dereference document fragments from the DOM tree. If an application processes the data in a deterministic order, then Oracle recommends avoiding the extra overhead of repeatedly releasing and re-creating nodes.

To use manual node dereferencing, set the attribute `PARTIAL_DOM` to `Boolean.FALSE` and create the SDOM with plug-in XML data.

To manually dereference a node from all other nodes, invoke `freeNode()`. For example:

```
Element root = doc.getDocumentElement();
 Node item = root.getFirstChild();
While (item != null)
{
     processItem(item);
     Node tmp = item;
     item = item.getNextSibling();
     ((XMLNode)tmp).freeNode();
}
```

Dereferencing a node does not remove it from the SDOM tree. The node can still be accessed and re-created from its parent, previous, and following siblings. However, after a node is dereferenced, a variable that holds the node throws an error when accessing the node.

Copyright © 1999, 2025, Oracle and/or its affiliates.

> **Note**
>
> The `freeNode` invocation has no effect on a nonscalable DOM.

12.3.4.2.4 Using Shadow Copy

Shadow copy avoids data replication by letting DOM nodes share their data.

Cloning, a common operation in XML processing, can be done lazily with SDOM. That is, the `copy` method creates only the root node of the fragment being copied, and the subtree is expanded only on demand.

DOM nodes themselves are not shared; their underlying data is shared. The DOM specification requires that the clone and its original have different node identities and different parent nodes.

12.3.4.2.5 Incorporating DOM Updates

The DOM API supports update operations such as adding and deleting nodes and setting, deleting, changing, and inserting values.

When a DOM is created by plugging in XML data, the underlying data is considered external to the DOM. DOM updates are visible from the DOM APIs but the data source remains the same. Normal update operations are available and do not interfere with each other.

To make a modified DOM persistent, you must explicitly save the DOM. Saving merges the changes with the original data and serializes the data in persistent storage. If you do not save a modified DOM explicitly, the changes are lost when the transaction ends.

12.3.4.2.6 Using the PageManager Interface to Support Internal Data

How to use the `PageManager` interface to let the SDOM use back-end storage for binary data.

When XML text is parsed with `DOMParser` and configured to create an SDOM, internal data is cached in the form of binary XML, and the DOM API layer is built on top of the internal data. This provides increased scalability, because the binary XML is more compact than DOM nodes.

For additional scalability, the SDOM can use back-end storage for binary data through the `PageManager` interface. Then, binary data can be swapped out of memory when not in use.

This code shows how to use the `PageManager` interface:

```
DOMParser parser = new DOMParser();
parser.setAttribute(PARTIAL_DOM, Boolean.TRUE); //enable SDOM
parser.setAttribute(PAGE_MANAGER, new FilePageManager("pageFile"));
...
// DOMParser other configuration
parser.parse(fileURL);
XMLDocument doc = parser.getDocument();
```

If you do not use the `PageManager` interface, then the parser caches the whole document as binary XML.

Copyright © 1999, 2025, Oracle and/or its affiliates.

12.3.4.3 Using Configurable DOM Settings

When you create a DOM using class `XMLDOMImplementation`, you can configure the DOM for different applications and achieve maximum efficiency by using method `setAttribute`.

```
public void setAttribute(String name, Object value) throws
IllegalArgumentException
```

For SDOM, invoke `setAttribute` for the `PARTIAL_DOM` and `ACCESS_MODE` attributes.

> **Note**
>
> New attribute values always affect the next DOM, not the current one. Therefore, you can use instances of `XMLDOMImplementation` to create DOMs with different configurations.

12.3.4.3.1 PARTIAL_DOM Attribute

Attribute `PARTIAL_DOM` determines whether the created DOM is partial, that is, scalable. When it has the value `TRUE`, the DOM is scalable (that is, nodes that are not in use are freed and re-created when needed). When it has the value `FALSE`, the created DOM is not scalable.

12.3.4.3.2 ACCESS_MODE Attribute

Attribute `ACCESS_MODE` (which applies to both SDOM and nonscalable DOM) controls access to the created DOM.

The attribute values, from least to most restrictive, are shown in Table 12-8.

Table 12-8 ACCESS_MODE Attribute Values

Value	DOM Access	Performance Advantage
`UPDATEABLE`	All update operations allowed. This is the default value, for backward compatibility with the XDK DOM implementation.	
`READ_ONLY`	No DOM update operations allowed. Node creation (for example, cloning) is allowed only if the new nodes are not added to the DOM tree.	Write buffer is not created.
`FORWARD_READ`	Forward navigation (for example, `getFirstChild().getNextSibling()` and `getLastChild()`) and access to parent and ancestor nodes is allowed; backward navigation (for example, `getPreviousSibling()`) is not allowed.	Previous-sibling links are not created.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 12-8 (Cont.) ACCESS_MODE Attribute Values

Value	DOM Access	Performance Advantage

STREAM_READ	Limited to the stream of nodes in document order, similar to SAX event access.
	The current node is the last node that was accessed in document order. Applications can hold nodes in variables and revisit them, but using the DOM method to access any node before the current node (except a parent or ancestor) causes an error. For example:
	• This is allowed, although the parent is before the current node:
	`Node parent = currentNode.getParentNode();`
	• This causes an error unless the current node is the first child of the parent:
	`Node child = parent.getFirstChild();`
	• Accessing element attributes is always allowed:
	`Attribute attr = parent.getFirstAttribute();`
		DOM maintains only parent links, not node locations; therefore, it need not re-create freed nodes.

12.3.4.4 Using Fast Infoset with SDOM

The Fast Infoset to XDK/J model lets you use Fast Infoset techniques while working with XML content in Java.

> **Note**
>
> Use Fast Infoset only for input. For output, use CSX or XTI.

This example uses a serializer to encode XML data into a FastInfoset `BinaryStream`:

```
public com.sun.xml.fastinfoset.sax.SAXDocumentSerializer getSAXDocumentSerializer();
public com.sun.xml.fastinfoset.stax.StAXDocumentSerializer getStAXDocumentSerializer();
```

The class `oracle.xml.scalable.BinaryStream` is the data management component that provides buffer management and an abstract paged I/O view to support decoding for different types of data storage.

The `InfosetReader` from `BinaryStream` is the implementation of `oracle.xml.scalable.InfosetReader` for the DOM to read data from binary. The implementation extends the basic decoder `sun.com.xml.fasterinfoset.Decoder` and adds support for seek and skip operations.

You can use Fast Infoset with Streaming API for XML (StAX) and SAX to create a DOM. To create an SDOM, you can use the routines from the preceding example and those in this example:

```
String xmlFile, fiFile;
FileInputStream xin = new FileInputStream(new File(xmlFile));
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
XML_SAX_FI figen = new XML_SAX_FI();
FileOutputStream outfi = new FileOutputStream(new File(fiFile));
figen.parse(xin, outfi);
outfi.close();

import oracle.xml.scalable.BinaryStream;

BinaryStream stream = BinaryStream.newInstance(SUN_FI);
stream.setFile(new File(fiFile));
InfosetReader reader = stream.getInfosetReader();
XMLDOMImplementation dimp = new  XMLDOMImplementation();
dimp.setAttribute(XMLDocument.SCALABLE_DOM, Boolean.TRUE);
XMLDocument doc = (XMLDocument) dimp.createDocument(reader);
```

12.3.4.5 SDOM Applications

Applications that create and use an SDOM are presented.

This application creates and uses an SDOM:

```
XMLDOMImplementation domimpl = new XMLDOMImplementation();
domimpl.setAttribute(XMLDocument.SCALABLE_DOM, Boolean.TRUE);
domimpl.setAttribute(XMLDocument.ACCESS_MODE,XMLDocument.UPDATEABLE);
XMLDocument scalableDoc = (XMLDocument) domimpl.createDocument(reader);
```

The following application creates and uses an SDOM based on binary XML, which is described in Using Binary XML with Java:

```
BinXMLProcessor proc = BinXMLProcessorFactory.createProcessor();
BinXMLStream bstr = proc.createBinXMLStream();
BinXMLEncoder enc = bstr.getEncoder();
enc.setProperty(BinXMLEncoder.ENC_SCHEMA_AWARE, false);

SAXParser parser = new SAXParser();
parser.setContentHandler(enc.getContentHandler());
parser.setErrorHandler(enc.getErrorHandler());
parser.parse(BinXMLUtil.createURL(xmlfile));

BinXMLDecoder dec = bstr.getDecoder();
InfosetReader reader = dec.getReader();
XMLDOMImplementation domimpl = new XMLDOMImplementation();
domimpl.setAttribute(XMLDocument.SCALABLE_DOM, Boolean.TRUE);
XMLDocument currentDoc = (XMLDocument) domimpl.createDocument(reader);
```

12.3.4.6 XDK Java DOM Improvements

XDK supports the DOM Level 3 Core specification, a recommendation of the W3C.

> **See Also**
>
> Document Object Model (DOM) Level 3 Core Specification for more information about DOM Level 3

Copyright © 1999, 2025, Oracle and/or its affiliates.

12.3.5 Performing DOM Operations with Namespaces

`DOM2Namespace.java` shows a simple use of the parser and namespace extensions to the DOM APIs. The program receives an XML document, parses it, and prints the elements and attributes in the document.

This section includes some code from the `DOM2Namespace.java` program. For more detail, see the program itself.

The first four steps of Performing Basic DOM Parsing, from parser creation to the `getDocument()` invocation, are basically the same for `DOM2Namespace.java`. The principal difference is in printing the DOM tree (Step 5). The `DOM2Namespace.java` program does this instead:

```
// Print document elements
printElements(doc);

// Print document element attributes
System.out.println("The attributes of each element are: ");
printElementAttributes(doc);
```

The `printElements()` method implemented by `DOM2Namespace.java` invokes `getElementsByTagName()` to get a list of all the elements in the DOM tree. It then loops through each item in the list and casts each `Element` to an `nsElement`. For each `nsElement` it invokes `nsElement.getPrefix()` to get the namespace prefix, `nsElement.getLocalName()` to get the local name, and `nsElement.getNamespaceURI()` to get the namespace URI:

```
static void printElements(Document doc)
{
   NodeList nl = doc.getElementsByTagName("*");
   Element nsElement;
   String prefix;
   String localName;
   String nsName;

   System.out.println("The elements are: ");
   for (int i=0; i < nl.getLength(); i++)
   {
      nsElement = (Element)nl.item(i);

      prefix = nsElement.getPrefix();
      System.out.println("  ELEMENT Prefix Name :" + prefix);

      localName = nsElement.getLocalName();
      System.out.println("  ELEMENT Local Name    :" + localName);

      nsName = nsElement.getNamespaceURI();
      System.out.println("  ELEMENT Namespace     :" + nsName);
   }
   System.out.println();
}
```

The `printElementAttributes()` method invokes `Document.getElementsByTagName()` to get a `NodeList` of the elements in the DOM tree. It then loops through each element and invokes `Element.getAttributes()` to get the list of attributes for the element as special list called a `NamedNodeMap`. For each item in the attribute list it invokes `nsAttr.getPrefix()` to get the namespace prefix, `nsAttr.getLocalName()` to get the local name, and `nsAttr.getValue()` to get the value:

Copyright © 1999, 2025, Oracle and/or its affiliates.

```
static void printElementAttributes(Document doc)
{
   NodeList nl = doc.getElementsByTagName("*");
   Element e;
   Attr nsAttr;
   String attrpfx;
   String attrname;
   String attrval;
   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 = (Attr) nnm.item(i);

            attrpfx = nsAttr.getPrefix();
            attrname = nsAttr.getLocalName();
            attrval = nsAttr.getNodeValue();

            System.out.println(" " + attrpfx + ":" + attrname + " = "
                               + attrval);
         }
      }
      System.out.println();
   }
}
```

12.3.6 Performing DOM Operations with Events

`EventSample.java` shows how to register events with an event listener. For example, adding a node to a specified DOM element triggers an event, which causes the listener to print information about the event.

This section includes some code from the `EventSample.java` program. For more detail, see the program itself.

The `EventSample.java` program follows these steps:

1. Instantiate an event listener.

   When a registered change triggers an event, the event is passed to the event listener, which handles it. This code fragment from `EventSample.java` shows the implementation of the listener:

   ```
   eventlistener evtlist = new eventlistener();
   ...
   class eventlistener implements EventListener
   {
      public eventlistener(){}
      public void handleEvent(Event e)
      {
   ```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
        String s = " Event "+e.getType()+" received " + "\n";
        s += " Event is cancelable :"+e.getCancelable()+"\n";
        s += " Event is bubbling event :"+e.getBubbles()+"\n";
        s += " The Target is " + ((Node)(e.getTarget())).getNodeName() + "\n\n";
        System.out.println(s);
    }
}
```

2. Instantiate a new `XMLDocument` and then invoke `getImplementation()` to retrieve a `DOMImplementation` object.

   Invoke the `hasFeature()` method to determine which features this implementation supports, as this code fragment from `EventSample.java` does:

```
XMLDocument doc1 = new XMLDocument();
DOMImplementation impl = doc1.getImplementation();

System.out.println("The impl supports Events "+
                   impl.hasFeature("Events", "2.0"));
System.out.println("The impl supports Mutation Events "+
                   impl.hasFeature("MutationEvents", "2.0"));
```

3. Register desired events with the listener. This code fragment from `EventSample.java` registers three events on the document node:

```
doc1.addEventListener("DOMNodeRemoved", evtlist, false);
doc1.addEventListener("DOMNodeInserted", evtlist, false);
doc1.addEventListener("DOMCharacterDataModified", evtlist, false);
```

   This code fragment from `EventSample.java` creates a node of type `XMLElement` and then registers three events on the node:

```
XMLElement el = (XMLElement)doc1.createElement("element");
...
el.addEventListener("DOMNodeRemoved", evtlist, false);
el.addEventListener("DOMNodeRemovedFromDocument", evtlist, false);
el.addEventListener("DOMCharacterDataModified", evtlist, false);
...
```

4. Perform actions that trigger events, which are then passed to the listener for handling, as this code fragment from `EventSample.java` does:

```
att.setNodeValue("abc");
el.appendChild(el1);
el.appendChild(text);
text.setNodeValue("xyz");
doc1.removeChild(el);
```

12.3.7 Performing DOM Operations with Ranges

According to the W3C DOM specification, a **range** identifies a range of content in a `Document`, `DocumentFragment`, or `Attr`. The range selects the content between a pair of boundary points that correspond to the start and end of the range.

Table 12-9 describes range methods accessible through `XMLDocument`.

Table 12-9 Range Class Methods

Method	Description
`cloneContents()`	Duplicates the contents of a range

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 12-9 (Cont.) Range Class Methods

Method	Description
`deleteContents()`	Deletes the contents of a range
`getCollapsed()`	Returns TRUE is the range is collapsed
`getEndContainer()`	Gets the node within which the range ends
`getStartContainer()`	Gets the node within which the range starts
`selectNode()`	Selects a node and its contents
`selectNodeContents()`	Selects the contents of a node
`setEnd()`	Sets the attributes describing the end of a range
`setStart()`	Sets the attributes describing the start of a range

The `DOMRangeSample.java` program shows some operations that you can perform with ranges. This section includes some code from the `DOMRangeSample.java` program. For more detail, see the program itself.

The first four steps of the Performing Basic DOM Parsing, from parser creation to the `getDocument()` invocation, are the same for `DOMRangeSample.java`. Then, the `DOMRangeSample.java` program follows these steps:

1. After invoking `getDocument()` to create the `XMLDocument`, create a range object with `createRange()` and invoke `setStart()` and `setEnd()` to set its boundaries, as this code fragment from `DOMRangeSample.java` does:

   ```
   XMLDocument doc = parser.getDocument();
   ...
   Range r = (Range) doc.createRange();
   XMLNode c = (XMLNode) doc.getDocumentElement();

   // set the boundaries
   r.setStart(c,0);
   r.setEnd(c,1);
   ```

2. Invoke `XMLDocument` methods to get information about the range and manipulate its contents.

   This code fragment from `DOMRangeSample.java` selects and prints the contents of the current node:

   ```
   r.selectNodeContents(c);
   System.out.println(r.toString());
   ```

   This code fragment clones and prints the contents of a range:

   ```
   XMLDocumentFragment df =(XMLDocumentFragment) r.cloneContents();
   df.print(System.out);
   ```

   This code fragment gets and prints the start and end containers for the range:

   ```
   c = (XMLNode) r.getStartContainer();
   System.out.println(c.getText());
   c = (XMLNode) r.getEndContainer();
   System.out.println(c.getText());
   ```


Copyright © 1999, 2025, Oracle and/or its affiliates.

12.3.8 Performing DOM Operations with TreeWalker

XDK implements the `NodeFilter` and `TreeWalker` interfaces, which are defined by the W3C DOM Level 2 Traversal and Range specification.

A node filter is an object that can filter out certain types of `Node` objects. For example, it can filter out entity reference nodes but accept element and attribute nodes. You create a node filter by implementing the `NodeFilter` interface and then passing a `Node` object to the `acceptNode()` method. Typically, the `acceptNode()` method implementation invokes `getNodeType()` to get the type of the node and compares it to static variables such as `ELEMENT_TYPE`, `ATTRIBUTE_TYPE`, and so forth, and then returns one of the static fields listed in Table 12-10, based on what it finds.

Table 12-10 Static Fields in the NodeFilter Interface

Field	Description
`FILTER_ACCEPT`	Accepts the node. Navigation methods defined for `NodeIterator` or `TreeWalker` return this node.
`FILTER_REJECT`	Rejects the node. Navigation methods defined for `NodeIterator` or `TreeWalker` do not return this node. For `TreeWalker`, the children of this node are also rejected. `NodeIterator` treats `FILTER_REJECT` as a synonym for `FILTER_SKIP`.
`FILTER_SKIP`	Skips this single node. Navigation methods defined for `NodeIterator` or `TreeWalker` do not return this node. For both `NodeIterator` and `TreeWalker`, children of this node are considered.

You can use a `TreeWalker` object to traverse a document tree or subtree, using the view of the document defined by the `whatToShow` flag and filters of the `TreeWalker` object.

To create a `TreeWalker` object, use the `XMLDocument.createTreeWalker()` method, specifying:

- A root node for the tree or subtree
- A flag that governs the type of nodes to include in the logical view
- A node filter (optional)
- A flag that determines whether to include entity references and their descendents

Table 12-11 describes methods in the `org.w3c.dom.traversal.TreeWalker` interface.

Table 12-11 TreeWalker Interface Methods

Method	Description
`firstChild()`	Moves the tree walker to the first visible child of the current node and returns the new node. If the current node has no visible children, then the method returns `null` and retains the current node.
`getRoot()`	Gets the root node of the tree walker (specified when the `TreeWalker` object was created).
`lastChild()`	Moves the tree walker to the last visible child of the current node and returns the new node. If the current node has no visible children, then the method returns `null` and retains the current node.
`nextNode()`	Moves the tree walker to the next visible node in document order relative to the current node and returns the new node.

Copyright © 1999, 2025, Oracle and/or its affiliates.

The `TreeWalkerSample.java` program shows some operations that you can perform with node filters and tree walkers. This section includes some code from the `TreeWalkerSample.java` program. For more detail, see the program itself.

The first four steps of the Performing Basic DOM Parsing, from parser creation to the `getDocument()` invocation, are the same for `TreeWalkerSample.java`. The, the `TreeWalkerSample.java` program follows these steps:

1. Create a node filter object.

   The `acceptNode()` method in the `nf` class, which implements the `NodeFilter` interface, invokes `getNodeType()` to get the type of node, as this code fragment from `TreeWalkerSample.java` does:

   ```
   NodeFilter n2 = new nf();
   ...
   class nf implements NodeFilter
   {
     public short acceptNode(Node node)
     {
       short type = node.getNodeType();

       if ((type == Node.ELEMENT_NODE) || (type == Node.ATTRIBUTE_NODE))
          return FILTER_ACCEPT;
       if ((type == Node.ENTITY_REFERENCE_NODE))
          return FILTER_REJECT;
       return FILTER_SKIP;
     }
   }
   ```

2. Invoke the `XMLDocument.createTreeWalker()` method to create a tree walker.

   This code fragment from `TreeWalkerSample.java` uses the root node of the `XMLDocument` as the root node of the tree walker and includes all nodes in the tree:

   ```
   XMLDocument doc = parser.getDocument();
   ...
   TreeWalker tw =
   doc.createTreeWalker(doc.getDocumentElement(),NodeFilter.SHOW_ALL,n2,true);
   ```

3. Get the root element of the `TreeWalker` object, as this code fragment from `TreeWalkerSample.java` does:

   ```
   XMLNode nn = (XMLNode)tw.getRoot();
   ```

4. Traverse the tree.

   This code fragment from `TreeWalkerSample.java` walks the tree in document order by invoking the `TreeWalker.nextNode()` method:

   ```
   while (nn != null)
   {
     System.out.println(nn.getNodeName() + " " + nn.getNodeValue());
     nn = (XMLNode)tw.nextNode();
   }
   ```

   This code fragment from `TreeWalkerSample.java` walks the left depth of the tree by invoking the `firstChild()` method:

   ```
    while (nn != null)
    {
      System.out.println(nn.getNodeName() + " " + nn.getNodeValue());
      nn = (XMLNode)tw.firstChild();
    }
   ```


Copyright © 1999, 2025, Oracle and/or its affiliates.

You can walk the right depth of the tree by invoking the `lastChild()` method.

12.4 Parsing XML with SAX

Simple API for XML (SAX) is a standard interface for event-based XML parsing.

12.4.1 Using the SAX API for Java

The interfaces and classes of the SAX API, which is released in a Level 1 and Level 2 version, are described.

These are the interfaces and classes:

- Interfaces implemented by the Oracle XML parser
- Interfaces that your application must implement (see Table 12-12)
- Standard SAX classes
- SAX 2.0 helper classes in the `org.xml.sax.helper` package (see Table 12-13)
- Demonstration classes in the `nul` package

Table 12-12 lists and describes the SAX 2.0 interfaces that your application must implement.

Table 12-12 SAX 2.0 Handler Interfaces

Interface	Description
`ContentHandler`	Receives notifications from the XML parser. Implements the major event-handling methods `startDocument()`, `endDocument()`, `startElement()`, and `endElement()`, which are invoked when the XML parser identifies an XML tag. Implements the methods `characters()` and `processingInstruction()`, which are invoked when the XML parser encounters the text in an XML element or an inline processing instruction.
`DeclHandler`	Receives notifications about DTD declarations in the XML document.
`DTDHandler`	Processes notations and unparsed (binary) entities.
`EntityResolver`	Supports redirection of URIs in documents. Implements the method `resolveEntity()`, which is invoked when the XML parser must identify data identified by a URI.
`ErrorHandler`	Handles parser errors. Implements the methods `error()`, `fatalError()`, and `warning()`, which the program invokes in response to various parsing errors.
`LexicalHandler`	Receives notifications about lexical information, such as comments and character data (CDATA) section boundaries.

Table 12-13 lists and describes the SAX 2.0 helper classes.

Table 12-13 SAX 2.0 Helper Classes

Class	Description
`AttributeImpl`	Makes a persistent copy of an `AttributeList`.
`DefaultHandler`	Base class with default implementations of the interfaces in Table 12-12.
`LocatorImpl`	Makes a persistent snapshot of the values of a Locator at a specified point in the parse.

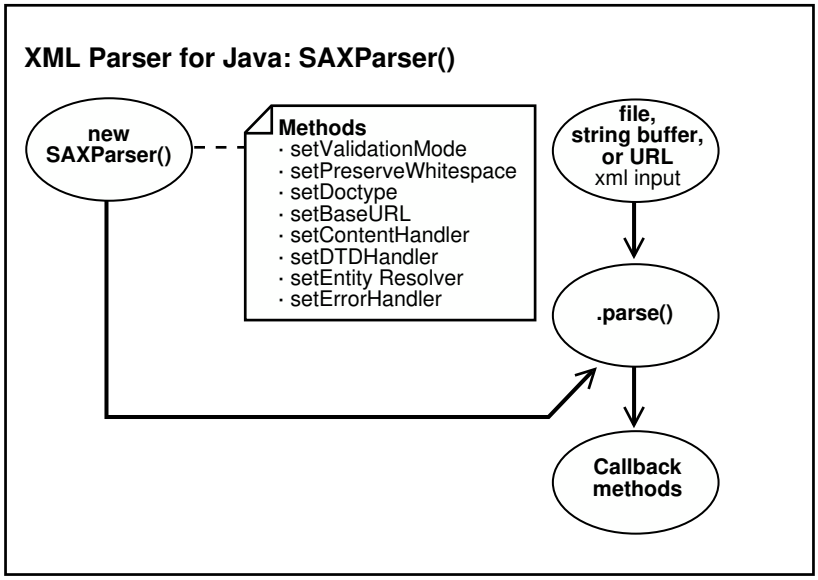
Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 12-13 (Cont.) SAX 2.0 Helper Classes

Class	Description
`NamespaceSupport`	Supports XML namespaces.
`XMLFilterImpl`	Base class used by applications that modify the stream of events.
`XMLReaderFactory`	Supports loading SAX parsers dynamically.

Figure 12-5 shows how to create a SAX parser and use it to parse an input document.

Figure 12-5 Using the SAXParser Class



The basic steps for parsing an input XML document with SAX are:

1. Create a `SAXParser` object and configure its properties.

   For example, set the validation mode. For configuration methods, see Table 12-5.

2. Instantiate an event handler.

   Your application must implement the handler interfaces in Table 12-12.

3. Register your event handlers with the XML parser.

   This step enables the parser to invoke the correct methods when a given event occurs. For information about `SAXParser` methods for registering event handlers, see Table 12-14.

4. Parse the input document with the `SAXParser.parse()` method.

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

   When the `SAXParser.parse()` method is invoked, the program invokes one of several callback methods implemented in the application. The methods are defined by the `ContentHandler`, `ErrorHandler`, `DTDHandler`, and `EntityResolver` interfaces implemented in the event handler. For example, the application can invoke the `startElement()` method when a start element is encountered.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 12-14 lists and describes the SAXParser methods for registering event handlers and explains when to use them. An application can register a new or different handler in the middle of a parse; the SAX parser must begin using the newly registered handler immediately.

Table 12-14 SAXParser Methods for Registering Event Handlers

Method	Description
`setContentHandler()`	Registers a content event handler with an application. The `org.xml.sax.DefaultHandler` class implements the `org.xml.sax.ContentHandler` interface.
`setDTDHandler()`	Registers a DTD event handler with an application. If the application does not register a DTD handler, DTD events reported by the SAX parser are silently ignored.
`setErrorHandler()`	Registers an error event handler with an application. If the application does not register an error handler, all error events reported by the SAX parser are silently ignored; however, normal processing may not continue. Oracle highly recommends that all SAX applications implement an error handler to avoid unexpected bugs.
`setEntityResolver()`	Registers an entity resolver with an application. If the application does not register an entity resolver, the `XMLReader` performs its own default resolution.

12.4.2 Performing Basic SAX Parsing

`SAXSample.java` shows the basic steps of SAX parsing. The `SAXSample` class extends `HandlerBase`. The program receives an XML file as input, parses it, and prints information about the contents of the file.

The `SAXSample.java` program follows these steps (which are illustrated with code fragments from the program):

1. Store the `Locator`:

   ```
   Locator locator;
   ```

   The `Locator` associates a SAX event with a document location. The SAX parser provides location information to the application by passing a `Locator` instance to the `setDocumentLocator()` method in the content handler. The application can use the object to get the location of any other content handler event in the XML source document.

2. Instantiate a new event handler.:

   ```
   SAXSample sample = new SAXSample();
   ```

3. Instantiate the SAX parser and configure it:

   ```
   Parser parser = new SAXParser();
   ((SAXParser)parser).setValidationMode(SAXParser.DTD_VALIDATION);
   ```

   The preceding code sets the mode to DTD validation.

4. Register event handlers with the SAX parser:

   ```
   parser.setDocumentHandler(sample);
   parser.setEntityResolver(sample);
   parser.setDTDHandler(sample);
   parser.setErrorHandler(sample);
   ```


Copyright © 1999, 2025, Oracle and/or its affiliates.

You can use the registration methods in the `SAXParser` class, but you must implement the event handler interfaces yourself.

Here is part of the `DocumentHandler` interface implementation:

```
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);
  }
}
...
```

The following code implements 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;
}
```

The following code implements 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);
}
```

The following code implements the `ErrorHandler` interface:

```
public void warning (SAXParseException e)
           throws SAXException
{
  System.out.println("Warning:"+e.getMessage());
}
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
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());
}
```

5. Parse the input XML document:

```
parser.parse(DemoUtil.createURL(argv[0]).toString());
```

The preceding code converts the document to a URL and then parses it.

12.4.3 Performing Basic SAX Parsing with Namespaces

`SAX2Namespace.java` implements an event handler named `XMLDefaultHandler` as a subclass of the `org.xml.sax.helpers.DefaultHandler` class.

The easiest way to implement the `ContentHandler` interface is to extend the `org.xml.sax.helpers.DefaultHandler` class. The `DefaultHandler` class provides some default behavior for handling events, although the typical behavior is to do nothing.

`SAX2Namespace.java` overrides methods only for relevant events. Specifically, the `XMLDefaultHandler` class implements only two methods: `startElement()` and `endElement()`. Whenever `SAXParser` encounters a new element in the XML document, it triggers the `startElement` event, and the `startElement()` method prints the namespace information for the element.

The `SAX2Namespace.java` sample program follows these steps (which are illustrated with code fragments from the program):

1. Instantiate a new event handler of type `DefaultHandler`:

```
DefaultHandler defHandler = new XMLDefaultHandler();
```

2. Create a SAX parser and set its validation mode:

```
Parser parser = new SAXParser();
((SAXParser)parser).setValidationMode(SAXParser.DTD_VALIDATION);
```

The preceding code sets the mode to DTD validation.

3. Register event handlers with the SAX parser:

```
parser.setContentHandler(defHandler);
parser.setEntityResolver(defHandler);
parser.setDTDHandler(defHandler);
parser.setErrorHandler(defHandler);
```

The preceding code registers handlers for the input document, the DTD, entities, and errors.

The following code shows the `XMLDefaultHandler` implementation. The `startElement()` and `endElement()` methods print the qualified name, local name, and namespace URI for each element (for an explanation of these terms, see Table 12-7).

```
class XMLDefaultHandler extends DefaultHandler
{
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
    public void XMLDefaultHandler(){}
    public void startElement(String uri, String localName,
                             String qName, Attributes atts)
    throws SAXException
    {
       System.out.println("ELEMENT Qualified Name:" + qName);
       System.out.println("ELEMENT Local Name    :" + localName);
       System.out.println("ELEMENT Namespace     :" + uri);

       for (int i=0; i<atts.getLength(); i++)
       {
          qName = atts.getQName(i);
          localName = atts.getLocalName(i);
          uri = atts.getURI(i);

          System.out.println(" ATTRIBUTE Qualified Name   :" + qName);
          System.out.println(" ATTRIBUTE Local Name       :" + localName);
          System.out.println(" ATTRIBUTE Namespace        :" + uri);

          // 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(String uri, String localName,
                           String qName) throws SAXException
    {
       System.out.println("ELEMENT Qualified Name:" + qName);
       System.out.println("ELEMENT Local Name    :" + localName);
       System.out.println("ELEMENT Namespace     :" + uri);
    }
}
```

4. Parse the input XML document:

```
parser.parse(DemoUtil.createURL(argv[0]).toString());
```

The preceding code converts the document to a URL and then parses it.

12.4.4 Performing SAX Parsing with XMLTokenizer

You can create a simple SAX parser as a instance of the `XMLTokenizer` class and use the parser to tokenize the input XML.

Table 12-15 lists useful methods in the class.

Table 12-15 XMLTokenizer Methods

Method	Description
`setToken()`	Registers a new token for XML tokenizer.
`setErrorStream()`	Registers a output stream for errors

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 12-15 (Cont.) XMLTokenizer Methods

Method	Description
`tokenize()`	Tokenizes the input XML

SAX parsers with `Tokenizer` features must implement the `XMLToken` interface. The callback method for `XMLToken` is `token()`, which receives an XML token and its corresponding value and performs an action. For example, you can implement `token()` so that it prints the token name followed by the value of the token.

The `Tokenizer.java` sample program accepts an XML document as input, parses it, and prints a list of the XML tokens. The program implements a `doParse()` method that follows these steps (which are illustrated with code fragments from the program):

1. Create a URL from the input XML stream:

   ```
   URL url = DemoUtil.createURL(arg);
   ```

2. Create an `XMLTokenizer` parser:

   ```
   parser  = new XMLTokenizer ((XMLToken)new Tokenizer());
   ```

3. Register an output error stream:

   ```
   parser.setErrorStream  (System.out);
   ```

4. Register tokens with the parser:

   ```
   parser.setToken (STagName, true);
   parser.setToken (EmptyElemTag, true);
   parser.setToken (STag, true);
   parser.setToken (ETag, true);
   parser.setToken (ETagName, true);
   ...
   ```

5. Tokenize the XML document:

   ```
   parser.tokenize (url);
   ```

   The `token()` callback method determines the action to take upon encountering a particular token. The following code is part of the implementation of this method:

   ```
   public void token (int token, String value)
   {
      switch (token)
      {
      case XMLToken.STag:
         System.out.println ("STag: " + value);
         break;
      case XMLToken.ETag:
         System.out.println ("ETag: " + value);
         break;
      case XMLToken.EmptyElemTag:
         System.out.println ("EmptyElemTag: " + value);
         break;
      case XMLToken.AttValue:
         System.out.println ("AttValue: " + value);
         break;
      ...
      default:
         break;
   ```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
        }
    }
```

12.5 Parsing XML with JAXP

JAXP lets your Java program use the SAX and DOM parsers and the XSLT processor.

12.5.1 JAXP Structure

JAXP consists of abstract classes that provide a thin layer for parser pluggability. Oracle implemented JAXP based on the Sun reference implementation.

Table 12-16 lists and describes the packages that comprise JAXP.

Table 12-16 JAXP Packages

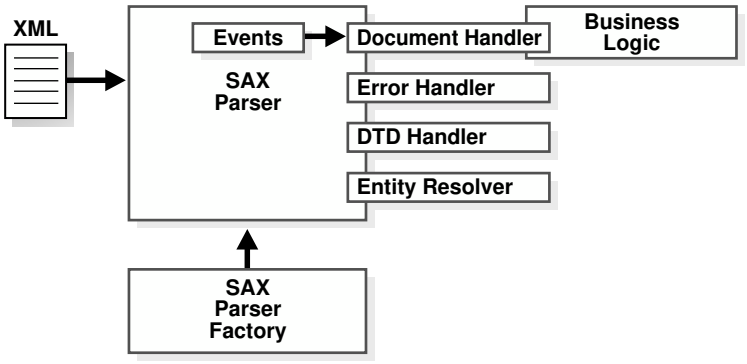
Package	Description
`javax.xml.parsers`	Provides standard APIs for DOM 2.0 and SAX 1.0 parsers. Contains vendor-neutral factory classes, including `SAXParser` and a `DocumentBuilder`. `DocumentBuilder` creates a DOM-compliant `Document` object.
`javax.xml.transform`	Defines the generic APIs for processing XML transformation and performing a transformation from a source to a result.
`javax.xml.transform.dom`	Provides DOM-specific transformation APIs.
`javax.xml.transform.sax`	Provides SAX2-specific transformation APIs.
`javax.xml.transform.stream`	Provides stream- and URI-specific transformation APIs.

12.5.2 Using the SAX API Through JAXP

You can rely on the factory design pattern to create new SAX parser engines with JAXP.

Figure 12-6 shows the basic process.

Figure 12-6 SAX Parsing with JAXP



The basic steps for parsing with SAX through JAXP are:

1. Create a new SAX parser factory with the `SAXParserFactory` class.

Copyright © 1999, 2025, Oracle and/or its affiliates.

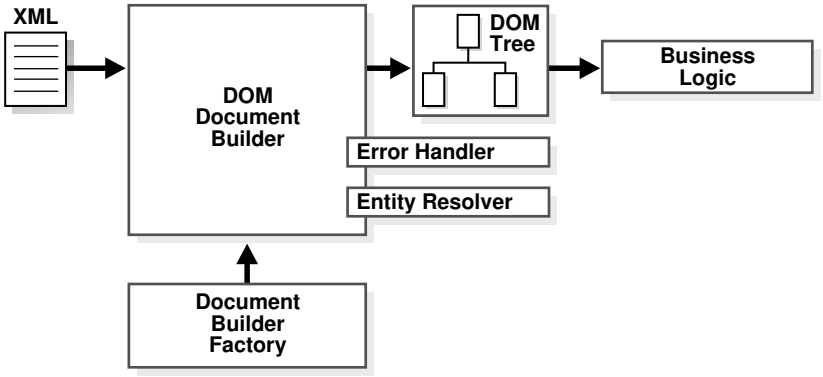
2. Configure the factory.
3. Create a new SAX parser (`SAXParser`) object from the factory.
4. Set the event handlers for the SAX parser.
5. Parse the input XML documents.

12.5.3 Using the DOM API Through JAXP

You can rely on the factory design pattern to create new DOM document builder engines with JAXP.

Figure 12-7 shows the basic process.

Figure 12-7 DOM Parsing with JAXP



The basic steps for parsing with DOM through JAXP are:

1. Create a new DOM parser factory with the `DocumentBuilderFactory` class.
2. Configure the factory.
3. Create a new DOM builder (`DocumentBuilder`) object from the factory.
4. Set the error handler and entity resolver for the DOM builder.
5. Parse the input XML documents.

12.5.4 Transforming XML Through JAXP

The basic steps for transforming XML through JAXP are described.

The steps are:

1. Create a new transformer factory with the `TransformerFactory` class.
2. Configure the factory.
3. Create a new transformer from the factory and specify an XSLT stylesheet.
4. Configure the transformer.
5. Transform the document.

Copyright © 1999, 2025, Oracle and/or its affiliates.

12.5.5 Parsing with JAXP

The `JAXPExamples.java` program shows the basic steps of parsing with JAXP.

The program implements these methods and uses them to parse and perform additional processing on XML files in the `/jaxp` directory:

- `basic()`
- `identity()`
- `namespaceURI()`
- `templatesHandler()`
- `contentHandler2contentHandler()`
- `contentHandler2DOM()`
- `reader()`
- `xmlFilter()`
- `xmlFilterChain()`

The program creates URLs for the sample XML files `jaxpone.xml` and `jaxpone.xsl` and then invokes the preceding methods in sequence. The basic design of the demo is as follows (to save space, only the `basic()` method is shown):

```
public class JAXPExamples
{
        public static void main(String argv[])
        throws TransformerException, TransformerConfigurationException,
               IOException, SAXException, ParserConfigurationException,
               FileNotFoundException
        {
        try {
         URL xmlURL = createURL("jaxpone.xml");
         String xmlID = xmlURL.toString();
         URL xslURL = createURL("jaxpone.xsl");
         String xslID = xslURL.toString();
         //
         System.out.println("--- basic ---");
         basic(xmlID, xslID);
         System.out.println();
         ...
      } catch(Exception err) {
        err.printStackTrace();
      }
   }
   //
   public static void basic(String xmlID, String xslID)
      throws TransformerException, TransformerConfigurationException
   {
      TransformerFactory tfactory = TransformerFactory.newInstance();
      Transformer transformer = tfactory.newTransformer(new StreamSource(xslID));
      StreamSource source = new StreamSource(xmlID);
      transformer.transform(source, new StreamResult(System.out));
   }
...
}
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

The `reader()` method in the program `JAXPExamples.java` shows a simple technique for parsing an XML document with SAX, using these steps (which are illustrated with code fragments from the program):

1. Create a new instance of a `TransformerFactory` and cast it to a `SAXTransformerFactory`:

   ```
   TransformerFactory tfactory = TransformerFactory.newInstance();
   SAXTransformerFactory stfactory = (SAXTransformerFactory)tfactory;
   ```

2. Create an XML reader by creating a `StreamSource` object from a stylesheet and passing it to the factory method `newXMLFilter()`:

   ```
   URL xslURL = createURL("jaxpone.xsl");
   String xslID = xslURL.toString();
   ...
   StreamSource streamSource = new StreamSource(xslID);
   XMLReader reader = stfactory.newXMLFilter(streamSource);
   ```

   `newXMLFilter()` returns an `XMLFilter` object that uses the specified `Source` as the transformation instructions.

3. Create a content handler and register it with the XML reader:

   ```
   ContentHandler contentHandler = new oraContentHandler();
   reader.setContentHandler(contentHandler);
   ```

   The preceding code creates an instance of the class `oraContentHandler` by compiling the `oraContentHandler.java` program in the demo directory.

   The following code shows part of the implementation of the `oraContentHandler` class:

   ```
   public class oraContentHandler implements ContentHandler
   {
      private static final String TRADE_MARK = "Oracle 9i ";

      public void setDocumentLocator(Locator locator)
      {
         System.out.println(TRADE_MARK + "- setDocumentLocator");
      }

      public void startDocument()
         throws SAXException
      {
         System.out.println(TRADE_MARK + "- startDocument");
      }

      public void endDocument()
         throws SAXException
      {
         System.out.println(TRADE_MARK + "- endDocument");
      }
      ...
   ```

4. Parse the input XML document by passing the `InputSource` to the `XMLReader.parse()` method:

   ```
   InputSource is = new InputSource(xmlID);
   reader.parse(is);
   ```


Copyright © 1999, 2025, Oracle and/or its affiliates.

12.5.6 Performing Basic Transformations with JAXP

You can use JAXP to perform basic transformations.

JAXP can transform these types of input:

- XML documents
- XSL stylesheets
- `ContentHandler` class defined in `oraContentHandler.java`

Here are some examples of using JAXP to perform basic transformations:

- You can use the `identity()` method to perform a transformation in which the output XML document and the input XML document are the same.
- You can use the `xmlFilterChain()` method to apply three stylesheets in a chain.
- You can transform any class of the interface `Source` into a class of the interface `Result` (`DOMSource` to `DOMResult`, `StreamSource` to `StreamResult`, `SAXSource` to `SAXResult`, and so on).

The `basic()` method in the program `JAXPExamples.java` shows how to perform a basic XSLT transformation, using these steps (which are illustrated with code fragments from the program):

1. Create a new instance of a `TransformerFactory`:

   ```
   TransformerFactory tfactory = TransformerFactory.newInstance();
   ```

2. Create a new XSL transformer from the factory and specify the stylesheet to use for the transformation:

   ```
   URL xslURL = createURL("jaxpone.xsl");
   String xslID = xslURL.toString();
   ...
   Transformer transformer = tfactory.newTransformer(new StreamSource(xslID));
   ```

   In the preceding code, the stylesheet is `jaxpone.xsl`.

3. Set the stream source to the input XML document:

   ```
   URL xmlURL = createURL("jaxpone.xml");
   String xmlID = xmlURL.toString();
   ...
   StreamSource source = new StreamSource(xmlID);
   ```

   In the preceding code, the stream source is `jaxpone.xml`.

4. Transform the document from a `StreamSource` to a `StreamResult`:

   ```
   transformer.transform(source, new StreamResult(System.out));
   ```

12.6 Compressing and Decompressing XML

XDK lets you use SAX or DOM to parse XML and then write the parsed data to a compressed binary stream. XDK also lets you reverse the process, decompressing the binary stream to reconstruct the XML data.

Copyright © 1999, 2025, Oracle and/or its affiliates.

12.6.1 Compressing a DOM Object

`DOMCompression.java` shows the basic steps of DOM compression. The most important DOM compression method is `XMLDocument.writeExternal()`, which saves the state of the object by creating a binary compressed stream with information about the object.

The `DOMCompression.java` program uses these steps (which are illustrated with code fragments from the program):

1. Create a DOM parser, parse an input XML document, and get the DOM representation:

```
public class DOMCompression
{
   static OutputStream out = System.out;
   public static void main(String[] args)
   {
      XMLDocument doc = new XMLDocument();
      DOMParser parser = new DOMParser();
      try
      {
        parser.setValidationMode(XMLParser.SCHEMA_VALIDATION);
        parser.setPreserveWhitespace(false);
        parser.retainCDATASection(true);
        parser.parse(createURL(args[0]));
        doc = parser.getDocument();
        ...
```

   For a description of this technique, see Performing Basic DOM Parsing.

2. Create a `FileOutputStream` and wrap it in an `ObjectOutputStream` for serialization:

```
OutputStream os = new FileOutputStream("xml.ser");
ObjectOutputStream oos = new ObjectOutputStream(os);
```

3. Serialize the object to the file by invoking `XMLDocument.writeExternal()`:

```
doc.writeExternal(oos);
```

   This method saves the state of the object by creating a binary compressed stream with information about this object.

12.6.2 Decompressing a DOM Object

`DOMDeCompression.java` shows the basic steps of DOM decompression. The most important DOM decompression method is `XMLDocument.readExternal()`, which reads the information that the `writeExternal()` method wrote (the compressed stream) and restores the object.

The `DOMDeCompression.java` program uses these steps (which are illustrated with code fragments from the program):

1. Create a file input stream for the compressed file and wrap it in an `ObjectInputStream`:

```
InputStream is;
ObjectInputStream ois;
...
is = new FileInputStream("xml.ser");
ois = new ObjectInputStream(is);
```

   The preceding code creates a `FileInputStream` from the compressed file created in Compressing a DOM Object.

Copyright © 1999, 2025, Oracle and/or its affiliates.

2. Create a new XML document object to contain the decompressed data:

```
XMLDocument serializedDoc = null;
serializedDoc = new XMLDocument();
```

3. Read the compressed file by invoking `XMLDocument.readExternal()`:

```
serializedDoc.readExternal(ois);
serializedDoc.print(System.out);
```

The preceding code data and prints it to `System.out`.

12.6.3 Compressing a SAX Object

`SAXCompression.java` shows the basic steps of parsing a file with SAX and writing the compressed stream to a file. The important class is `CXMLHandlerBase`, which is a SAX `Handler` that compresses XML data based on SAX events.

To use SAX compression, implement this interface and register it with the SAX parser by invoking `Parser.setDocumentHandler()`.

The `SAXCompression.java` program uses these steps (which are illustrated with code fragments from the program):

1. Create a `FileOutputStream` and wrap it in an `ObjectOutputStream`:

```
String compFile = "xml.ser";
FileOutputStream outStream = new FileOutputStream(compFile);
ObjectOutputStream out = new ObjectOutputStream(outStream);
```

2. Create the SAX event handler:

```
CXMLHandlerBase cxml = new CXMLHandlerBase(out);
```

The `CXMLHandlerBase` class implements the `ContentHandler`, `DTDHandler`, `EntityResolver`, and `ErrorHandler` interfaces.

3. Create the SAX parser:

```
SAXParser parser = new SAXParser();
```

4. Configure the SAX parser:

```
parser.setContentHandler(cxml);
parser.setEntityResolver(cxml);
parser.setValidationMode(XMLConstants.NONVALIDATING);
```

The preceding code sets the content handler, entity resolver, and validation mode.

> **Note**
>
> Although `oracle.xml.comp.CXMLHandlerBase` implements both `DocumentHandler` and `ContentHandler` interfaces, Oracle recommends using the SAX 2.0 `ContentHandler` interface.

5. Parse the XML:

```
parser.parse(url);
```

The `SAXCompression.java` program writes the serialized data to the `ObjectOutputStream`.

Copyright © 1999, 2025, Oracle and/or its affiliates.

12.6.4 Decompressing a SAX Object

`SAXDeCompression.java` shows the basic steps of reading the serialized data from the file that `SAXCompression.java` wrote. The important class is `CXMLParser`, which is an XML parser that regenerates SAX events from a compressed stream.

The `SAXDeCompression.java` program follows these steps (which are illustrated with code fragments from the program):

1. Create a SAX event handler:

   ```
   SampleSAXHandler xmlHandler = new SampleSAXHandler();
   ```

2. Create the SAX parser by instantiating the `CXMLParser` class:

   ```
   CXMLParser parser = new CXMLParser();
   ```

   The `CXMLParser` class implements the regeneration of XML documents from a compressed stream by generating SAX events from them.

3. Set the event handler for the SAX parser:

   ```
   parser.setContentHandler(xmlHandler);
   ```

4. Parse the compressed stream and generate the SAX events:

   ```
   parser.parse(args[0]);
   ```

   The preceding code receives a file name from the command line and parses the XML.

12.7 Tips and Techniques for Parsing XML

A few parsing tips and techniques are listed.

12.7.1 Extracting Node Values from a DOM Tree

You can use the `selectNodes()` method in the `XMLNode` class to extract content from a DOM tree or subtree based on the select patterns allowed by XSL.

You can use the optional second parameter of `selectNodes()` to resolve namespace prefixes; that is, to return the expanded namespace URL when given a prefix. The `XMLElement` class implements `NSResolver`, so a reference to an `XMLElement` object can be sent as the second parameter. `XMLElement` resolves the prefixes based on the input document. You can use the `NSResolver` interface to override the namespace definitions.

The sample code in Example 12-4 shows how to use `selectNodes()`.

To test the program, create a file with the code in Example 12-4, and then compile it in the `$ORACLE_HOME/xdk/demo/java/parser/common` directory. Pass the file name `family.xml` to the program as a parameter to traverse the `<family>` tree. The output is similar to this:

```
% java selectNodesTest family.xml
Sarah
Bob
Joanne
Jim
```

Now run the following code to determine the values of the `memberid` attributes of all `<member>` elements in the document:

Copyright © 1999, 2025, Oracle and/or its affiliates.

```
% java selectNodesTest family.xml //member/@memberid
m1
m2
m3
m4
```

Example 12-4 Extracting Contents of a DOM Tree with selectNodes()

```
//
// selectNodesTest.java
//
import java.io.*;
import oracle.xml.parser.v2.*;
import org.w3c.dom.Node;
import org.w3c.dom.Element;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;

public class selectNodesTest
{
  public static void main(String[] args)
    throws Exception
  {
    // supply an xpath expression
    String pattern = "/family/member/text()";
    // accept a filename on the command line
    // run the program with $ORACLE_HOME/xdk/demo/java/parser/common/family.xml
    String file    = args[0];

    if (args.length == 2)
      pattern = args[1];

    DOMParser dp = new DOMParser();

    dp.parse(DemoUtil.createURL(file));  // include createURL from DemoUtil
    XMLDocument xd = dp.getDocument();
    XMLElement element = (XMLElement) xd.getDocumentElement();
    NodeList nl = element.selectNodes(pattern, element);
    for (int i = 0; i < nl.getLength(); i++)
    {
      System.out.println(nl.item(i).getNodeValue());
    } // end for
  } // end main
} // end selectNodesTest
```

12.7.2 Merging Documents with appendChild()

How to merge XML documents using `XMLElement.appendChild()` is described.

To write a program that lets a user complete a client-side Java form and get an XML document, your Java program can contain these variables:

```
String firstname = "Gianfranco";
String lastname = "Pietraforte";
```

To insert this information into an XML document, you can use either of these techniques:

- Create an XML document in a string and then parse it. For example:

```
String xml = "<person><first>"+firstname+"</first>"+
     "<last>"+lastname+"</last></person>";
DOMParser d = new DOMParser();
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
d.parse(new StringReader(xml));
Document xmldoc = d.getDocument();
```

- Use DOM APIs to construct an XML document, creating elements and then appending them to one another. For example:

```
Document xmldoc = new XMLDocument();
Element e1 = xmldoc.createElement("person");
xmldoc.appendChild(e1);
Element e2 = xmldoc.createElement("firstname");
e1.appendChild(e2);
Text t = xmldoc.createText("Larry");
e2.appendChild(t);
```

You can use the second technique only on a *single* DOM tree.

Example 12-5 uses two trees—the owner document of `e1` is `xmldoc1` and the owner document of `e2` is `xmldoc2`. The `appendChild()` method works only within a single tree. Therefore, invoking `XMLElement.appendChild()` raises a DOM exception of `WRONG_DOCUMENT_ERR`.

To copy and paste a DOM document fragment or a DOM node across different XML documents, use the `XMLDocument.importNode()` method (introduced in DOM 2) and the `XMLDocument.adoptNode()` method (introduced in DOM 3). The comments in Example 12-6 show this technique.

Example 12-5 Incorrect Use of appendChild()

```
XMLDocument xmldoc1 = new XMLDocument();
XMLElement e1 = xmldoc1.createElement("person");
XMLDocument xmldoc2 = new XMLDocument();
XMLElement e2 = xmldoc2.createElement("firstname");
e1.appendChild(e2);
```

Example 12-6 Merging Documents with appendChild

```
XMLDocument doc1 = new XMLDocument();
XMLElement element1 = doc1.createElement("person");
XMLDocument doc2 = new XMLDocument();
XMLElement element2 = doc2.createElement("firstname");
// element2 = doc1.importNode(element2);
// element2 = doc1.adoptNode(element2);
element1.appendChild(element2);
```

12.7.3 Parsing DTDs

You can use load and parse a DTD.

12.7.3.1 Loading External DTDs

The procedure for loading and parsing a DTD is presented.

If you invoke the `DOMParser.parse()` method to parse the XML document as an `InputStream`, then use the `DOMParser.setBaseURL()` method to recognize external DTDs within your Java program. `DOMParser.setBaseURL()` points to a location where the DTDs are exposed.

The procedure for loading and parsing a DTD is:

1. Load the DTD as an `InputStream`.

   For example, this code validates documents against the `/mydir/my.dtd` external DTD:

   ```
   InputStream is = MyClass.class.getResourceAsStream("/mydir/my.dtd");
   ```


Copyright © 1999, 2025, Oracle and/or its affiliates.

The preceding code opens `./mydir/my.dtd` in the first relative location in the `CLASSPATH` where it can be found, including the JAR file if it is in the `CLASSPATH`.

2. Create a DOM parser and set the validation mode.

   For example:

   ```
   DOMParser d = new DOMParser();
   d.setValidationMode(DTD_VALIDATION);
   ```

3. Parse the DTD.

   For example, this code passes the `InputStream` object to the `DOMParser.parseDTD()` method:

   ```
   d.parseDTD(is, "rootelementname");
   ```

4. Get the document type and then set it.

   For example, in this code, the `getDoctype()` method gets the DTD object and the `setDoctype()` method sets the DTD to use for parsing:

   ```
   d.setDoctype(d.getDoctype());
   ```

   Alternatively, you can invoke the `parseDTD()` method to parse a DTD file separately and get a DTD object:

   ```
   d.parseDTD(new FileReader("/mydir/my.dtd"));
   DTD dtd = d.getDoctype();
   parser.setDoctype(dtd);
   ```

5. Parse the input XML document:

   ```
   d.parse("mydoc.xml");
   ```

12.7.3.2 Caching DTDs with setDoctype

The XML parser for Java provides for DTD caching in validation and nonvalidation modes through the `DOMParser.setDoctype()` method. After you set the DTD with this method, the parser caches it for further parsing.

> **Note**
>
> DTD caching is optional, and is not enabled automatically.

Suppose that your program must parse several XML documents with the same DTD. After you parse the first XML document, you can get the DTD from the parser and set it. For example:

```
DOMParser parser = new DOMParser();
DTD dtd = parser.getDoctype();
parser.setDoctype(dtd);
```

Example 12-7 invokes `DOMParser.setDoctype()` to cache the DTD.

If the cached DTD object is used only for validation, then set the `DOMParser.USE_DTD_ONLY_FOR_VALIDATION` attribute:

```
parser.setAttribute(DOMParser.USE_DTD_ONLY_FOR_VALIDATION,Boolean.TRUE);
```

Otherwise, the XML parser copies the DTD object and adds it to the resulting DOM tree.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Example 12-7 DTDSample.java

```
/**
 * DESCRIPTION
 * This program illustrates DTD caching.
 */

import java.net.URL;
import java.io.*;
import org.xml.sax.InputSource;
import oracle.xml.parser.v2.*;

public class DTDSample
{
   static public void main(String[] args)
   {
      try
      {
         if (args.length != 3)
         {
            System.err.println("Usage: java DTDSample dtd rootelement xmldoc");
            System.exit(1);
         }

         // Create a DOM parser
         DOMParser parser = new DOMParser();

         // Configure the parser
         parser.setErrorStream(System.out);
         parser.showWarnings(true);

        // Create a FileReader for the DTD file specified on the command
        // line and wrap it in an InputSource
        FileReader r = new FileReader(args[0]);
        InputSource inSource = new InputSource(r);

        // Create a URL from the command-line argument and use it to set the
        // system identifier
        inSource.setSystemId(DemoUtil.createURL(args[0]).toString());

        // Parse the external DTD from the input source. The second argument is
        // the name of the root element.
        parser.parseDTD(inSource, args[1]);
        DTD dtd = parser.getDoctype();

        // Create a FileReader object from the XML document specified on the
        // command line
        r = new FileReader(args[2]);

        // Wrap the FileReader in an InputSource,
        // create a URL from the filename,
        // and set the system identifier
        inSource = new InputSource(r);
        inSource.setSystemId(DemoUtil.createURL(args[2]).toString());

        // ********************
        parser.setDoctype(dtd);
        // ********************

        parser.setValidationMode(DOMParser.DTD_VALIDATION);
       // parser.setAttribute
       //   (DOMParser.USE_DTD_ONLY_FOR_VALIDATION,Boolean.TRUE);
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
            parser.parse(inSource);

            // Get the DOM tree and print
            XMLDocument doc = parser.getDocument();
            doc.print(new PrintWriter(System.out));

          }
          catch (Exception e)
          {
             System.out.println(e.toString());
          }
       }
}
```

12.7.4 Handling Character Sets with the XML Parser

Topics for handling character sets with the parser are introduced.

12.7.4.1 Detecting the Encoding of an XML File on the Operating System

Use the XML parser to detect the character encoding of an XML file stored on your file system.

When reading an XML file stored on the operating system, do not use the `FileReader` class. Instead, use the XML parser to detect the character encoding of the document automatically. Given a binary `FileInputStream` with no external encoding information, the parser automatically determines the character encoding based on the byte-order mark and encoding declaration of the XML document. You can parse any well-formed document in any supported encoding with the sample code in the `AutoDetectEncoding.java` demo, which is located in `$ORACLE_HOME/xdk/demo/java/parser/dom`.

> **Note**
>
> Include the proper encoding declaration in your document, according to the specification. `setEncoding()` cannot set the encoding for your input document. `setEncoding()` is used with `oracle.xml.parser.v2.XMLDocument` to set the correct encoding for printing.

12.7.4.2 Preventing Distortion of XML Stored in an NCLOB Column

To avoid distortion of XML data that is stored in an `NCLOB` column, use methods `getUnicodeStream()` and `getBinaryStream()`, or print the data to ensure that its characters are not distorted before they are sent to the parser.

Suppose that you load XML into a national character large object (`NCLOB`) column of a database using 8-bit encoding of Unicode (UTF-8), and the XML contains two UTF-8 multibyte characters:

```
G(0xc2,0x82)otingen, Br(0xc3,0xbc)ck_W
```

You write a Java stored function that does this:

1. Uses the default connection object to connect to the database.
2. Runs a `SELECT` query.
3. Gets the `oracle.jdbc.OracleResultSet` object.

Copyright © 1999, 2025, Oracle and/or its affiliates.

4. Invokes the `OracleResultSet.getCLOB()` method.
5. Invokes the `getAsciiStream()` method on the `CLOB` object.
6. Executes this code to get the XML into a DOM object:

```
DOMParser parser = new DOMParser();
parser.setPreserveWhitespace(true);
parser.parse(istr);
// istr getAsciiStream XMLDocument xmldoc = parser.getDocument();
```

The program throws an exception stating that the XML contains an invalid UTF-8 encoding even though the character (`0xc2`, `0x82`) is valid UTF-8. The problem is that the character can be distorted when the program invokes the `OracleResultSet.getAsciiStream()` method. To solve this problem, invoke the `getUnicodeStream()` and `getBinaryStream()` methods instead of `getAsciiStream()`. If this technique does not work, then try to print the characters to ensure that they are not distorted before they are sent to the parser when you invoke `DOMParser.parse(istr)`.

12.7.4.3 Writing an XML File in a Nondefault Encoding

A technique is introduced to avoid problems that can be introduced when writing XML files that contain characters that are not available in the default character encoding.

UTF-8 encoding is popular for XML documents, but UTF-8 is not usually the default file encoding of Java. Using a Java class in your program that assumes the default file encoding can cause problems.

For example, the Java class `FileWriter` depends on the default character encoding of the runtime environment. If you use the `FileWriter` class when writing XML files that contain characters that are not available in the default character encoding, then the output file can suffer parsing errors or data loss.

To avoid such problems, use the technique shown in the `I18nSafeXMLFileWritingSample.java` program in `$ORACLE_HOME/xdk/demo/java/parser/dom`.

You cannot use `System.out.println()` to output special characters. You must use a binary output stream that is encoding-aware, such as `OutputStreamWriter`. Construct an `OutputStreamWriter` and use the `write(char[], int, int)` method to print, as in this example:

```
/* Java encoding string for ISO8859-1*/
OutputStreamWriter out = new OutputStreamWriter(System.out, "8859_1");
OutputStreamWriter.write(...);
```

12.7.4.4 Parsing XML Stored in Strings

To parse an XML document contained in a `String`, you must first convert the string to an `InputStream` or `InputSource` object.

Example 12-8 converts a string of XML (referenced by `xmlDoc`) to a byte array, converts the byte array to a `ByteArrwayInputStream`, and then parses it.

You can convert the `XMLDocument` object created in the previous code back to a string by wrapping a `StringWriter` in a `PrintWriter`. This example shows this technique:

To convert the `XMLDocument` object created in Example 12-8 back to a string, you can wrap a `StringWriter` in a `PrintWriter`:

```
StringWriter sw = new StringWriter();
PrintWriter  pw = new PrintWriter(sw);
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
doc.print(pw);
String YourDocInString = sw.toString();
```

`ParseXMLFromString.java`, which is located in `$ORACLE_HOME/xdk/demo/java/parser/dom`, is a complete program that creates an XML document as a string and parses it.

Example 12-8 Converting XML in a String

```
// create parser
DOMParser parser=new DOMParser();
// create XML document in a string
String xmlDoc =
       "<?xml version='1.0'?>"+
       "<hello>"+
       "  <world/>"+
       "</hello>";
// convert string to bytes to stream
byte aByteArr [] = xmlDoc.getBytes();
ByteArrayInputStream bais = new ByteArrayInputStream(aByteArr,0,aByteArr.length);
//  parse and get DOM tree
DOMParser.parse(bais);
XMLDocument doc = parser.getDocument();
```

12.7.4.5 Parsing XML Documents with Accented Characters

Tips for parsing XML documents that contain accented characters are presented.

Example 12-9 shows one way to parse an XML document with accented characters (such as é).

When you try to parse the XML file, the parser might throw an "Invalid UTF-8 encoding" exception. The encoding is a scheme used to write the Unicode character number representation to disk. If you explicitly set the encoding to UTF-8 or do not specify the encoding, then the parser interprets an accented character—which has an ASCII value greater than 127—as the first byte of a UTF-8 multibyte sequence. If the subsequent bytes do not form a valid UTF-8 sequence, then you get an error.

The error means that your XML editor did not save the file with UTF-8 encoding. The editor might have saved the file with ISO-8859-1 (Western European ASCII) encoding. Adding the following element to the top of an XML document does not cause your editor to write the bytes representing the file to disk with UTF-8 encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
```

One solution is to read accented characters in their hexadecimal or decimal format within the XML document; for example, `Ù`. If you prefer not to use this technique, then you can set the encoding based on the character set that you were using when you created the XML file (for example, ISO-8859-1).

Example 12-9 Parsing a Document with Accented Characters

```
DOMParser parser=new DOMParser();
parser.setPreserveWhitespace(true);
parser.setErrorStream(System.err);
parser.setValidationMode(false);
parser.showWarnings(true);
parser.parse (new FileInputStream(new File("file_with_accents.xml")));
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

12.7.4.6 Handling Special Characters in Tag Names

Tips for handling special characters in XML element names are presented.

If a tag (element) name contains special characters (`&`, `$`, and `#`, and so on), then the parser issues an error about invalid characters.

If you are creating a new XML document, choose tag names that have no invalid `NameChar` characters. For example, if you want to name the tags after companies, and one company has the name A&B, then instead of the invalid tag `<A&B>`, choose `<A_B>`, `<AB>`, or `<A_AND_B>`.

If you are generating XML from external data sources such as database tables, then:

- XML 1.0 does not address this problem.
- In XML 1.1, the data type `XMLType` addresses this problem by providing the `setConvertSpecialChars` and `convert` functions in the `DBMS_XMLGEN` package.

  You can use these functions to control the use of special characters in structured query language (SQL) names and XML names. The SQL-to-XML name-mapping functions escape invalid XML `NameChar` characters in the format of *`_XHHHH_`*, where *`HHHH`* is the Unicode value of the invalid character. For example, table name `V$SESSION` is mapped to XML name `V_X0024_SESSION`.

  Escaping invalid characters provides a way to serialize names so that they can be reloaded somewhere else.

Copyright © 1999, 2025, Oracle and/or its affiliates.

13

Using Binary XML with Java

Topics here explain how to use Binary XML with Java.

13.1 Introduction to Binary XML for Java

Binary XML makes it possible to encode and decode between XML text and compressed binary XML. Application programming interfaces (APIs) are provided on top of Binary XML for direct consumption by the XML applications. Compression and decompression of fragments of an XML document facilitate incremental processing.

This chapter assumes that you are familiar with the XML Parser for Java.

Related Topics

- XML Parsing for Java
  Extensible Markup Language (XML) parsing for Java is described.

13.1.1 Binary XML Storage Format – Java

Binary XML is a compact XML-Schema-aware encoding of XML data, but it can be used with XML data that is not based on an XML schema. You can also use binary XML for XML data which is outside the database (in a client-side application, for instance). Binary XML allows for encoding and decoding of XML documents, from text to binary and binary to text. Binary XML is post-parse persistent XML with native database data types.

`XMLType` tables and columns can be created using the binary XML storage option. The XML data in binary format can be accessed and manipulated by all the existing structured query language (SQL) operators and functions and Procedural Language/Structured Query Language (PL/SQL) APIs that operate on `XMLType`.

Binary XML provides more efficient database storage, updating, indexing, query performance, and fragment extraction than unstructured storage. It can store data and metadata together, or separately.

> **See Also**
>
> *Oracle XML DB Developer's Guide* for a discussion of all the storage models in Oracle XML DB.

13.1.2 Binary XML Processors

A *binary XML processor* is an abstract term for a component that processes and transforms binary XML into text and XML text into binary XML. It can also provide a cache for storing schemas. A binary XML processor can originate or receive network protocol requests.

The base class for a binary XML processor is `BinXMLProcessor`.

Copyright © 1999, 2025, Oracle and/or its affiliates.

13.2 Models for Using Binary XML

There are several models for using binary XML in applications. These subsections describe the terminology and the models for using binary XML.

13.2.1 Usage Terminology for Binary XML

Terms related to binary XML usage are described.

- *doc-id*: Each encoded XML document is identified by a unique doc-id. It is either a 16-byte Global User identifier (GUID) or an opaque sequence of bytes like a URL.
- *token table*: When a text XML document does not have a schema associated with it, then a token (or symbol) table is used to minimize space for repeated items.
- *vocabulary id*: Can be a schema-id or a namespace Universal Resource Identifier (URI) for a token table.
- *schema-id*: A unique opaque binary identifier for a schema scoped to the binary XML processor. The schema-id is unique for a binary XML processor and is identifiable only within the scope of that binary XML processor. The schema-id remains constant even when the schema is evolved. A schema-id represents the entire set of schema documents, including imported and included schemas.
- *schema version*: Every annotated schema has a version number associated with it. The version number is specified as part of the system level annotations. It is incremented by the binary XML processor when a schema is evolved (that is, a new version of the same schema is registered with the binary XML processor).
- *partial validity*: Binary XML stream encoding using schema implies at least partial validity with the schema. Partial validity implies no validation for unique keys, keyrefs, identifiers (IDs), or DTD attributes such as IDREF.

13.2.2 Standalone Model

This is the simplest usage scenario for binary XML. There is a single binary XML processor.

The only repository available is the local in-memory vocabulary cache that is not persistent and is available only for the life of the binary XML processor. All schemas must be registered in advance with the binary XML Processor before the encoding, or they can be registered automatically when the XML Processor sees the `xsi:SchemaLocation` tag. For decoding, the schema is already available in the vocabulary cache.

13.2.3 Client/Server Model

In a client-server scenario, the binary XML processor is connected to a database using Java Database Connectivity (JDBC). It is assumed that the XML schema is registered with the database before encoding.

Here is an example of how to achieve that:

```
BEGIN
   DBMS_XMLSCHEMA.registerSchema(
   SCHEMAURL =>
   'http://xmlns.oracle.com/xdb/documentation/purchaseOrder.xsd',
   SCHEMADOC =>
   bfilename('XMLDIR','purchaseOrder.xsd'),
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
   CSID => nls_charset_id('AL32UTF8'),
   GENTYPES => FALSE,
   OPTIONS => REGISTER_BINARYXML );
END;
/
```

Unless a separate connection is specified for data (using `associateDataConnection()`) it is assumed that all data and metadata is stored and retrieved using a single connection for encoding and decoding.

13.2.4 Web Services Model With Repository

In this scenario there are multiple clients, each running a binary XML processor. One encodes and another decodes. There is a common repository (that is not necessarily a database) connected to all the clients for metadata storage. It can be a file system or some other repository.

The first binary XML processor ensures that the schema is registered with the repository before performing the encoding, or the schema might be automatically registered using the `xsi:schemaLocation` tag at the time of encoding. The second binary XML processor is used for decoding, is not aware of the location of the schema, and fetches the schema from the repository.

If the first binary XML processor registers a schema and the second binary XML processor registers the same schema in the repository, the binary XML processor does not compile the schema, but simply returns the `vocabulary-id` of the existing compiled schema in the local vocabulary cache.

The `BinXMLProcessor` is not thread-safe, so multiple threads or clients accessing the repository must implement their own thread safety scheme.

13.2.5 Web Services Model Without Repository

In this scenario, there are multiple clients, each running a binary XML processor. Encoding and decoding can happen on different clients. There is no common metadata repository.

The encoder must ensure that the binary data passed to the next client is independent of schema: that is, has inline token definitions. This can be achieved by setting `schemaAware = false` and `inlineTokenDefs = true`, using the `setProperty()` method, during encoding. While decoding, there is no schema required.

13.3 Components of Binary XML for Java

The components of binary XML for Java are described.

These are the components:

- Binary XML encoding—The binary XML encoder converts XML 1.0 infoset to binary XML.
- Binary XML decoding—The binary XML decoder converts binary XML to XML infoset.
- Binary XML vocabulary management, which includes schema management and token management.

Copyright © 1999, 2025, Oracle and/or its affiliates.

13.3.1 Binary XML Encoding

The encoder is created from a `BinXMLStream`. It takes XML text as input, and it outputs the encoded binary XML to the `BinXMLStream`. It reads the XML text using streaming SAX. The encoding of the XML text is based on the results of parsing the XML text.

Set the `schemaAware` flag on the encoder that specifies whether the encoding is schema-aware or schema-less.

For schema-aware encoding, the encoder determines whether the schema with the specified schema URL has been registered with the vocabulary manager. For a repository-based or a database-based processor, the encoder queries the repository or the database for the compiled schema based on the schema URL. If the schema is available in the database, it is fetched from the repository or database in the binary XML format and registered with the local vocabulary manager. The vocabulary is schema.

Also set a flag to indicate that the encoding produces a binary XML stream that is independent of a schema. In this case, the resulting binary XML stream contains all token definitions inline and is not dependent on schema or external token sets.

If the encoding is schema-aware, the encoder uses the data type information from the schema object for more efficient encoding of the SAX stream. There is a default encoding data type associated with each schema built-in data type. Binary XML stream encoding using a schema implies at least partial validity with the schema (For partial validity there is no validation for unique key, or keyref, or ID, or DTD attributes such as IDREF). If the data is known to be completely valid with a schema, the encoded binary XML stream stores this information.

> **See Also**
>
> *Oracle XML DB Developer's Guide* for tables of the binary encoding data types and their mappings from XML schema data types

If there is no schema associated with the text XML, then integer token ids are generated for repeated items in the text XML. Creating a token table of token ids and token definitions is an important compression technique. The token definitions are stored as token tables in the vocabulary cache. If the property for inline token definitions is set, then the token definitions are present inline.

Another property on the encoder is specifying PSVI (Post-Schema-Validated Infoset) information as part of the binary stream. If this is set to true then PSVI information can be accessed using XDK extension APIs for PSVI on DOM. If `psvi = true` then the input XML is fully validated with the schema. If `psvi` is `false` then PSVI information is not included in the output binary stream. The default is `false`.

Related Topics

- Token Management
  Token sets can be fetched from the database or metadata repository, cached in the local vocabulary manager, and used for decoding. While encoding, token sets can be pushed to the repository for persistence.

Copyright © 1999, 2025, Oracle and/or its affiliates.

13.3.2 Binary XML Decoding

The binary XML decoder converts binary XML to XML infoset. The decoder is created from the `BinXMLStream`; it reads binary XML from this stream and outputs SAX events or provide a pull style `InfosetReader` API for reading the decoded XML.

If an XML schema is associated with the `BinXMLStream`, the binary XML decoder retrieves the associated schema object from the vocabulary cache, using the vocabulary id before decoding. If the schema is not available in the vocabulary cache and the connection information to the server is available, then the schema is fetched from the server.

If no schema is associated with `BinXMLStream`, then the token definitions can be either inline in the `BinXMLStream` or stored in a token set. If tokens of a corresponding namespace are not stored in the local vocabulary cache, then the token set is fetched from the repository.

13.4 Binary XML Vocabulary Management

The binary XML processors are of different types depending on where the metadata (schema or token sets) are located—either local binary XML processor or repository binary XML processor.

13.4.1 Schema Management

For metadata persistence, Oracle recommends that you use the DB Binary XML processor. In this case, schemas and token sets are registered with the database. The vocabulary manager fetches the schema or token sets from the database and cache it in the local vocabulary cache for encoding and decoding.

If you must use a persistent metadata repository that is not a database, you can plug in your own metadata repository. You must implement the interface for communicating with this repository, `BinXMLMetadataProvider`.

Related Topics

- Binary XML
  A binary XML processor can communicate with the database for various types of binary XML operations involving storage and retrieval of binary XML schemas, token sets, and binary XML streams.

13.4.1.1 Schema Registration for Binary XML Vocabulary Management

Register XML schemas locally with the local binary XML processor. It contains a vocabulary manager that maintains all XML schemas submitted by a user for the duration of its existence. The vocabulary manager associated with a local binary XML processor does not provide for XML schema persistence.

If you register the same XML schema again (same schema location and same target namespace) then it is not parsed, and the existing vocabulary identifier is returned.

If a new XML schema with the same target namespace and a different schema location is registered, then the existing XML schema definition is augmented with the new schema definitions. In case of conflict, an error is raised.

Copyright © 1999, 2025, Oracle and/or its affiliates.

13.4.1.2 Schema Identification

Each schema is identified by a vocabulary id. The vocabulary id is in the scope of the processor and is unique within the processor. Any document that validates with a schema is required to validate with a latest version of the schema.

13.4.1.3 Schema Annotations

Binary XML Schema annotations can appear only within element `<xsd:appInfo>` in an XML schema. The vocabulary manager interprets user-level and system-level annotations during XML schema registration. All other schema annotations, such as database-related annotations, are ignored.

13.4.1.4 User-Level Annotations

User-level annotations are specified by a user before registration.

`encodingType`—This annotation can be used within a `xsd:element`, `xsd:attribute` or `xsd:simpleType` elements. It indicates the data type to be used for encoding the node value of the element or attribute. For strings, there is support only for 8-bit encoding of Unicode (UTF-8) encoding in this release.

13.4.1.5 System-Level Annotations

The vocabulary manager adds system-level annotations at the time of registration. You cannot overwrite them.

13.4.2 Token Management

Token sets can be fetched from the database or metadata repository, cached in the local vocabulary manager, and used for decoding. While encoding, token sets can be pushed to the repository for persistence.

Token definitions can also be included as part of the binary XML stream by setting a flag on the encoder.

13.5 Using the Java Binary XML Package

Use of the binary XML package is described.

A `BinXMLStream` class represents the binary XML stream. The different storage locations defined for the binary XML stream are:

- `InputStream`—stream for reading.
- `OutputStream`—stream for writing.
- `URL`—stream for reading.
- `File`—stream for read and write.
- `BLOB`—stream for reading and writing.
- Byte array—stream for reading and writing.
- In memory—stream for reading and writing.

Copyright © 1999, 2025, Oracle and/or its affiliates.

The `BinXMLStream` object specifies the type of storage during creation.

A `BinXMLStream` object can be created from a `BinXMLProcessor` factory. This factory can be initialized with a JDBC connection (for remote metadata access), connection pool, URL or a `PageManagerPool` (for lazy in-memory storage). `BinXMLEncoder` and `BinXMLDecoder` can be created from the `BinXMLStream` for encoding or decoding.

Here is an example of creating a processor without a repository, registering a schema, encoding XML SAX events into schema-aware binary format, and storing in a file:

```
BinXMLProcessor  proc = BinXMLProcessorFactory.createProcessor();
proc.registerSchema(schemaURL);
BinXMLStream outbin = proc.createBinaryStream(outFile);
BinXMLEncoder enc = outbin.getEncoder();
enc.setSchemaAware(true);
ContentHandler hdlr = enc.getContentHandler();
```

In addition to getting the `ContentHandler`, you can also get the other handlers, such as:

```
LexicalHandler lexhdlr = enc.getLexicalHandler();
DTDHandler dtdhdlr = encenc.getDTDHandler();
DeclHandler declhdlr = enc.getDeclHandler();
ErrorHandler errhdlr = enc.getErrorHandler();
```

Use `hdlr` in the application that generates the SAX events.

2. Here is an example of creating a processor with a database repository, decoding a schema-aware binary stream and reading the decoded XML using pull API. The schema is fetched from the database repository for decoding.

```
DBBinXMLMetadataProvider dbrep =
     BinXMLMetadataProviderFactory.createDBMetadataProvider();
BinXMLProcessor proc = BinXMLProcessorFactory.createProcessor(dbrep);
BinXMLStream inpbin = proc.createBinaryStream(blob);
BinXMLDecoder dec = inpbin.getDecoder();
InfosetReader xmlreader = dec.getReader();
```

Use `xmlreader` to read XML in a pull-style from the decoder.

13.5.1 Binary XML Encoder

The encoder takes XML input, which is parsed and read using SAX events, and outputs binary XML.

13.5.1.1 Schema-Less Option

You can specify the schema-aware or the schema-less option before encoding. The default is schema-less encoding.

If the schema-aware option is set, then the encoding is done based on schema(s) specified in the instance document. The annotated schemas used for encoding are also required at the time of decoding. If the schema-less option is specified, then the encoding is independent of schemas, but the tokens are inline by default. To override the default, set `Inline-token = false`.

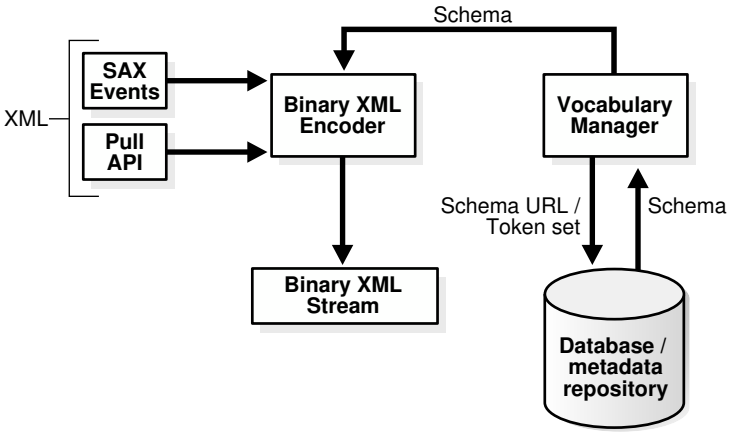
Copyright © 1999, 2025, Oracle and/or its affiliates.

13.5.1.2 Inline-Token Option

You can set an option to create a binary XML stream with inline token definitions before encoding. If inlining is turned off then you must ensure that the encoder or decoder processors use the same metadata repository. By default, token definition is inline.

Flag `Inline-token` is ignored if the schema-aware option is turned on.

Figure 13-1 Binary XML Encoding

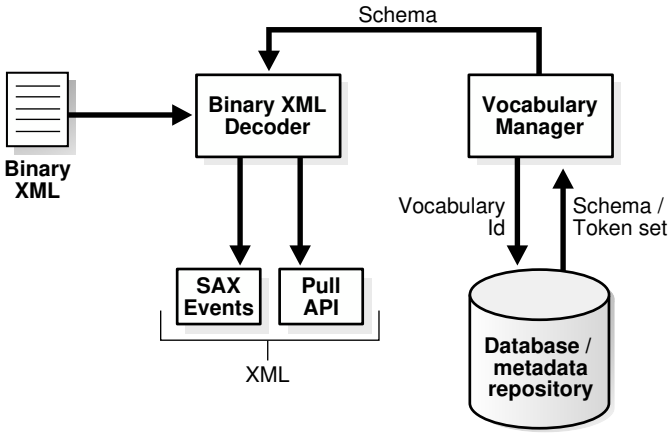


13.5.2 Binary XML Decoder

The binary XML decoder takes a binary XML stream as input and generates SAX Events as output, or it provides a pull interface to read the decoded XML. For an XML schema-aware binary XML stream, the binary XML decoder interacts with the vocabulary manager to extract the schema information.

If the vocabulary manager does not contain the required schema, and the processor is of type binary XML with a valid JDBC connection, then the remote schema is fetched from the database or the metadata repository based on the vocabulary id in the binary XML stream to be decoded. Similarly, the set of token definitions can be fetched from the database or the metadata repository.

Figure 13-2 Binary XML Decoder



Copyright © 1999, 2025, Oracle and/or its affiliates.

13.5.3 Schema Registration Overview

You register an XML schema with the Binary XML Processor. The schema is a text file that can contain user-level annotations. As part of the registration process, the processor adds system-level annotations. The resulting annotated schema is then processed by the Schema Builder to build an XML schema object.

This XML schema object is stored in the vocabulary cache. The vocabulary cache assigns a unique vocabulary id for each XML schema object, which is returned as output. The annotated DOM representation of the XML schema is sent to the binary XML encoder.

13.5.4 Resolving xsi:schemaLocation

How `xsi:schemaLocation` is resolved is described.

During encoding, if `schemaAware` is true and the property `ImplcitSchemaRegistration` is `true`, then the first `xsi:schemaLocation` tag present in the root element of an XML instance document automatically registers that schema in the local vocabulary manager. No other `schemaLocation` tags are explicitly registered. If the processor is database-oriented then the schema is also registered in the database; similarly for any metadata repository based processor.

If the encoding is set to `schemaAware` is `false` or `ImplcitSchemaRegistration` is `false`, then all `xsi:schemaLocation` tags are ignored by the encoder.

13.5.5 Binary XML

A binary XML processor can communicate with the database for various types of binary XML operations involving storage and retrieval of binary XML schemas, token sets, and binary XML streams.

A `DBBinXMLMetadataProvider` object is either instantiated with a dedicated JDBC connection or a connection pool to access vocabulary information such as schema and token set. The processor is also associated with one or more data connections to access XML data.

Database communication is involved in these ways:

1. Extracting compiled binary XML schema using the vocabulary ID or the schema URL

   To retrieve a compiled binary XML schema for encoding, the database is queried based on the schema URL. For decoding the binary XML schema, fetch it from the database based on the vocabulary ID.

2. Storing noncompiled binary XML schema using the schema URL and retrieving the vocabulary id.

   When the `xsi:schemaLocation` tag is encountered during encoding, the schema is registered in the database for persistent storage in the database. The vocabulary id associated with the schema, and the binary version of the compiled schema is retrieved from the database; the compiled schema object is built and stored in the local cache using the vocabulary id returned from the database.

3. Retrieving a binary token set using namespace URL.

   If a binary stream to be decoded is associated with token tables for decoding, these are fetched from the database using the metadata connection.

4. Storing binary token set using namespace URL

Copyright © 1999, 2025, Oracle and/or its affiliates.

If the XML text has been encoded without a schema, then it produces a token set of token definitions. These token tables can be stored persistently in the database. The metadata connection is used for transferring the token set to the database.

5. Binary XML stream with remote storage option

   It is your responsibility to create a table containing an `XMLType` column with binary XML for storing the result of encoding and retrieving the binary XML for decoding. Communication with the database can be achieved with Oracle Net Services and JDBC. Fetch the `XMLType` object from the output result set of the JDBC query. The `BinXMLStream` for reading the binary data or for writing out binary data can be created from the `XMLType` object. The `XMLType` class must be extended to support reading and writing of binary XML data.

13.5.6 Persistent Storage of Metadata

You can provide persistent back-end storage for metadata.

A local vocabulary manager and cache stores metadata information in memory for the life of the `BinXMLProcessor`. You can plug in your own back-end storage for metadata by implementing interface `BinXMLMetadataProvider` and plugging it into the `BinXMLProcessor`. Currently only one metadata provider for each processor is supported.

You must code a `FileBinXMLMetadataProvider` that implements interface `BinXMLMetadataProvider`. The encoder and decoder uses these APIs to access metadata from the persisted back-end storage. Set up the configuration information for the persistent storage: for example, root directory for a file system in `FileBinXMLMetadataProvider` class. Instantiate `FileBinXMLMetadataProvider` and plug it into the `BinXMLProcessor`.

Copyright © 1999, 2025, Oracle and/or its affiliates.

14

Using the XSLT Processor for Java

An explanation is given of how to use the Extensible Stylesheet Language Transformation (XSLT) processor for Java.

14.1 Introduction to the XSLT Processor

Topics include prerequisites, standards and specifications, and an overview of XML transformation with XSLT.

14.1.1 Prerequisites for Using the XSLT Processor for Java

XSLT is a language, based on Extensible Markup Language (XML), that you can use to transform one XML document into another text document. For example, you can use XSLT to accept an XML data document as input, perform arithmetic calculations on element values in the document, and generate an Extensible HyperText Markup Language (XHTML) document that shows the calculation results.In XSLT, XPath is used to navigate and process elements in the source node tree. XPath models an XML document as a tree made up of nodes; the types of nodes in the XPath node tree correspond to the types of nodes in a DOM tree.

This chapter assumes that you are familiar with these World Wide Web Consortium (W3C) standards:

- Extensible Stylesheet Language (XSL) and Extensible Stylesheet Language Transformations (XSLT). For a general introduction to XSLT, see the XML resources listed in Related Documents.

14.1.2 Standards and Specifications for the XSLT Processor for Java

The Oracle XML Developer's Kit (XDK) XSLT processor supports the XSLT2.0 recommendation.

XPath, which is the navigational language used by XSLT and other XML languages, is available in two versions: XPath 2.0 and the XPath 1.0 Recommendation.

Related Topics

- Oracle XML Developer's Kit Standards
  A description is given of the Oracle XML Developer's Kit (XDK) standards.

> **See Also**
>
> - XSL Transformations (XSLT) Version 2.0
> - XML Path Language (XPath) 2.0
> - XML Path Language (XPath)

Copyright © 1999, 2025, Oracle and/or its affiliates.

14.1.3 XML Transformation with XSLT 1.0 and 2.0

Oracle XML Developer's Kit (XDK) provides several useful features not included in XSLT 1.0. To use XSLT 2.0, set the `version` attribute in your stylesheet.

```
<? xml-stylesheet version="2.0" ... ?>
```

Useful XSLT 2.0 features include these:

- User-defined functions

  You can use the `<xsl:function>` declaration to define functions. This element must have one `name` attribute to define the function name. The value of the `name` attribute is a `QName`. The content of the `<xsl:function>` element is zero or more `xsl:param` elements that specify the formal arguments of the function, followed by a sequence constructor that defines the value returned by the function.

  `QName` can have a null namespace, but user-defined functions must have a non-null namespace. That is, if `abc` is defined as a namespace, then `add` is not a legal user-defined function, but `abc:add` is.

- Grouping

  You can use the `<xsl:for-each-group>` element, `current-group()` function, and `current-grouping-key()` function to group items.

- Multiple result documents

  You can use the `<xsl:result-document>` element to create a result tree. The content of the `<xsl:result-document>` element is a sequence constructor for the children of the document node of the tree.

  For example, this element enables you to accept an XML document as input and break it into separate documents. You can take an XML document that describes a list of books and generate an XHTML document for each book. You can then validate each output document.

- Temporary trees

  Instead of representing the intermediate XSL transformation results and XSL variables as strings, as in XSLT 1.0, you can store them as a set of document nodes. The document nodes, which you can construct with the `<xsl:variable>`, `<xsl:param>`, and `<xsl:with-param>` elements, are called temporary trees.

- Character mapping

  In XSLT 1.0, you had to use the `disable-output-escaping` attribute of the `<xsl:text>` and `<xsl:value-of>` elements to specify character escaping. In XSLT 2.0, you can declare mapping characters with an `<xsl:character-map>` element as a top-level `stylesheet` element. You can use this element to generate files with reserved or invalid XML characters in the XSLT outputs, such as <, >, and &.

> **See Also**
>
> XSL Transformations (XSLT) Version 2.0 for explanation and examples of XSLT 2.0 features

Copyright © 1999, 2025, Oracle and/or its affiliates.

14.2 Using the XSLT Processor for Java: Overview

The XDK XSLT processor transforms an XML document into another text-based document, with a format such as XML, HTML, XHTML, or plain text. You can invoke the processor programmatically by using an application programming interface (API) or run it from the command line.

The XSLT processor can perform these tasks:

- Reads one or more XSLT stylesheets. The processor can apply multiple stylesheets to a single XML input document and generate different results.
- Reads one or more input XML documents. The processor can use a single stylesheet to transform multiple XML input documents.
- Builds output documents by applying the rules in the stylesheet to the input XML documents. The output is a Document Object Model (DOM) tree, output stream, or series of Simple API for XML (SAX) events.

Whereas XSLT is a function-based language that generally requires a DOM of the input document and stylesheet to perform the transformation, the XDK Java implementation of the XSLT processor can use SAX to create a stylesheet object to perform transformations with higher efficiency and fewer resources. You can reuse this stylesheet object to transform multiple documents without reparsing the stylesheet.

14.2.1 Using the XSLT Processor for Java: Basic Process

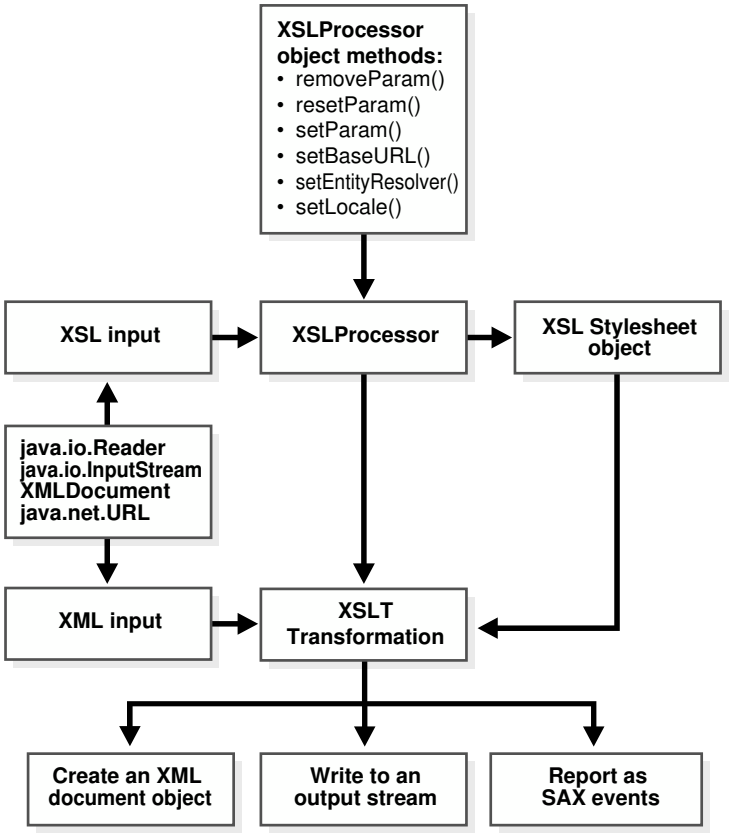
The basic design of the XSLT processor for Java is presented.

Figure 14-1 illustrates this process.

> **See Also**
>
> *Oracle AI Database XML Java API Reference* to learn about the `XMLParser` and `XSDBuilder` classes

Copyright © 1999, 2025, Oracle and/or its affiliates.

Figure 14-1 Using the XSLT Processor for Java



14.2.2 Running the XSLT Processor Demo Programs

Demo programs for the XSLT processor for Java are included in `$ORACLE_HOME/xdk/demo/java/parser/xslt`.

Table 14-1 describes the XML files and programs that you can use to test the XSLT processor.

Table 14-1 XSLT Processor Sample Files

File	Description
`match.xml`	A sample XML document that you can use to test ID selection and pattern matching. Its associated stylesheet is `match.xsl`.
`match.xsl`	A sample stylesheet for use with `match.xml`. You can use it to test simple identity transformations.
`math.xml`	A sample XML data document that you can use to perform simple arithmetic. Its associated stylesheet is `math.xsl`.
`math.xsl`	A sample stylesheet for use with `math.xml`. The stylesheet outputs an HTML page with the results of arithmetic operations performed on element values in `math.xml`.
`number.xml`	A sample XML data document that you can use to test for source tree numbering. The document describes the structure of a book.
`number.xsl`	A sample stylesheet for us with `number.xml`. The stylesheet outputs an HTML page that calculates section numbers for the sections in the book described by `number.xml`.
`position.xml`	A sample XML data document that you can use to test for `position()=X` in complex patterns. Its associated stylesheet is `position.xsl`.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 14-1 (Cont.) XSLT Processor Sample Files

File	Description
`position.xsl`	A sample stylesheet for use with `position.xml`. The stylesheet outputs an HTML page with the results of complex pattern matching.
`reverse.xml`	A sample XML data document that you can use with `reverse.xsl` to traverse backward through a tree.
`reverse.xsl`	A sample stylesheet for us with `reverse.xml`. The stylesheet output the item numbers in `reverse.xml` in reverse order.
`string.xml`	A sample XML data document that you can use to test perform various string test and manipulations. Its associated stylesheet is `string.xsl`.
`string.xsl`	A sample stylesheet for us with `string.xml`. The stylesheet outputs an XML document that displays the results of the string manipulations.
`style.txt`	A stylesheet that provides the framework for an HTML page. The stylesheet is included by `number.xsl`.
`variable.xml`	A sample XML data document that you can use to test the use of XSL variables. The document describes the structure of a book. Its associated stylesheet is `variable.xsl`.
`variable.xsl`	A stylesheet for use with `variable.xml`. The stylesheet makes extensive use of XSL variables.
`XSLSample.java`	A sample application that offers a simple example of how to use the XSL processing capabilities of the Oracle XSLT processor. The program transforms an input XML document by using an input stylesheet. This program builds the result of XSL transformations as a `DocumentFragment` and does not show `xsl:output` features. Run this program with any XSLT stylesheet in the directory as a first argument and its associated `*.xml` XML document as a second argument. For example, run the program with `variable.xsl` and `variable.xml` or `string.xsl` and `string.xml`.
`XSLSample2.java`	A sample application that offers a simple example of how to use the XSL processing capabilities of the Oracle XSLT processor. The program transforms an input XML document by using an input stylesheet. This program outputs the result to a stream and supports `xsl:output` features. Like `XSLSample.java`, you can run it against any pair of XML data documents and stylesheets in the directory.

Documentation for how to compile and run the sample programs is located in the `README`. The basic steps are:

1. Change into the `$ORACLE_HOME/xdk/demo/java/parser/xslt` directory (UNIX) or `%ORACLE_HOME%\xdk\demo\java\parser\xslt` directory (Windows).
2. Make sure that your environment variables are set as described in Setting Up the XDK for Java Environment
3. Run `make` (UNIX) or `Make.bat` (Windows) at the command line. The make file compiles the source code and then runs the `XSLSample` and `XSLSample2` programs for each `*.xml` file and its associated `*.xsl` stylesheet. The program writes its output for each transformation to `*.out`.
4. You can view the `*.out` files to see the output for the XML transformations. You can also run the programs on the command line as follows, where *`name`* is replaced by `match`, `math`, and so forth:

   ```
   java XSLSample name.xsl name.xml
   java XSLSample2 name.xsl name.xml
   ```

   For example, run the `match.xml` demos:

Copyright © 1999, 2025, Oracle and/or its affiliates.

```
java XSLSample match.xsl match.xml
java XSLSample2 match.xsl match.xml
```

14.2.3 Using the XSLT Processor Command-Line Utility

XDK includes `oraxsl`, which is a command-line Java interface that can apply a stylesheet to multiple XML documents. The `$ORACLE_HOME/bin/oraxsl` and `%ORACLE_HOME%\bin\oraxsl.bat` shell scripts execute the `oracle.xml.jaxb.oraxsl` class.

To use `oraxsl` ensure that your `CLASSPATH` is set as described in Setting Up the XDK for Java Environment.

Use this syntax on the command line to invoke `oraxsl`:

`oraxsl` *`options source stylesheet result`*

The `oraxsl` utility expects a stylesheet, an XML file to transform, and an optional result file. If you do not specify a result file, then the utility sends the transformed document to standard output. If multiple XML documents must be transformed by a stylesheet, then use the `-l` or `-d` options with the `-s` and `-r` options. These and other options are described in Table 14-2.

Table 14-2 Command-Line Options for oraxsl

Option	Description
`-w`	Shows warnings. By default, warnings are turned off.
`-e` *`error_log`*	Specifies file into which the program writes errors and warnings.
`-l` *`xml_file_list`*	Lists files to be processed.
`-d` *`directory`*	Specifies the directory that contains the files to transform. The default behavior is to process all files in the directory. If only a subset of the files in that directory, for example, one file, must be processed, then change this behavior by setting `-l` and specifying the files that must be processed. You can also change the behavior by using the `-x` or `-i` option to select files based on their extension.
`-x` *`source_extension`*	Specifies extensions for the files to be excluded. Use this option with `-d`. The program does not select any files with the specified extension.
`-i` *`source_extension`*	Specifies extensions for the files to be included. Use this option with `-d`. The program selects only files with the specified extension.
`-s` *`stylesheet`*	Specifies the stylesheet. If you set `-d` or `-l`, then set `-s` to indicate the stylesheet to be used. You must specify the complete path.
`-r` *`result_extension`*	Specifies the extension to use for results. If you set `-d` or `-l`, then set `-r` to specify the extension to be used for the results of the transformation. So, if you specify the extension `out`, the program transformed an input document `doc` to `doc.out`. By default, the program places the results in the current directory. You can change this behavior by using the `-o` option, which enables you to specify a directory for the results.
`-o` *`result_directory`*	Specifies the directory in which to place results. You must set this option with the `-r` option.
`-p` *`param_list`*	Lists parameters.
`-t` *`num_of_threads`*	Specifies the number of threads to use for processing. Using multiple threads can provide performance improvements when processing multiple documents.
`-v`	Generates verbose output. The program prints some debugging information and can help in tracing any problems that are encountered during processing.
`-debug`	Generates debugging output. By default, debug mode is disabled. A graphical user interface (GUI) version of the XSLT debugger is available in Oracle JDeveloper.

Copyright © 1999, 2025, Oracle and/or its affiliates.

14.2.3.1 Using the XSLT Processor Command-Line Utility: Example

You can test `oraxsl` on the various XML files and stylesheets in `$ORACLE_HOME/xdk/demo/java/parser/xslt`.

Example 14-1 displays the contents of `math.xml`.

The XSLT stylesheet named `math.xsl` is shown in Example 14-2.

You can run the `oraxsl` utility on these files to produce HTML output as shown in this example:

```
oraxsl math.xml math.xsl math.htm
```

The output file `math.htm` is shown in Example 14-3.

Example 14-1 math.xml

```
<?xml version="1.0"?>
<doc>
  <n1>5</n1>
  <n2>2</n2>
  <div>-5</div>
  <mod>2</mod>
</doc>
```

Example 14-2 math.xsl

```
<?xml version="1.0"?><xsl:stylesheet version="1.0" xmlns:xsl="http://
www.w3.org/1999/XSL/Transform">
  <xsl:template match="doc">
    <HTML>
      <H1>Test for mod.</H1>
      <HR/>
      <P>Should say "1": <xsl:value-of select="5 mod 2"/></P>
      <P>Should say "1": <xsl:value-of select="n1 mod n2"/></P>
      <P>Should say "-1": <xsl:value-of select="div mod mod"/></P>
      <P><xsl:value-of select="div or ((mod)) | or"/></P>
    </HTML>
  </xsl:template>
</xsl:stylesheet
```

Example 14-3 math.htm

```
<HTML>
   <H1>Test for mod.</H1>
   <HR>
   <P>Should say "1": 1</P>
   <P>Should say "1": 1</P>
   <P>Should say "-1": -1</P>
   <P>true</P>
</HTML>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

14.3 Transforming XML

Topics here include performing basic XSL transformation and getting DOM results from a transformation.

14.3.1 Performing Basic XSL Transformation

The fundamental classes used by the XSLT processor are `DOMParser` and `XSLProcessor`. The `XSL2Sample.java` demo program provides a good illustration of how to use these classes to transform an XML document with an XSLT stylesheet.

Classes `DOMParser` and `XSLProcessor` are described in Using the XSLT Processor for Java: Overview.

Use these basic steps to write Java programs that use the XSLT processor:

1. Create a DOM parser object that you can use to parse the XML data documents and XSLT stylesheets. This code fragment from `XSL2Sample.java` shows how to instantiate a parser:

   ```
   XMLDocument xml, xsldoc, out;URL xslURL;URL xmlURL;
   // ...
   parser = new DOMParser();parser.setPreserveWhitespace(true);
   ```

   By default, the parser does not preserve white space unless a DTD is used. It is important to preserve white space because it enables XSLT white space rules to determine how white space is handled.

2. Parse the XSLT stylesheet with the `DOMParser.parse()` method. this code fragment from `XSL2Sample.java` shows how to perform the parse:

   ```
   xslURL = DemoUtil.createURL(args[0]);
   parser.parse(xslURL);
   xsldoc = parser.getDocument();
   ```

3. Parse the XML data document with the `DOMParser.parse()` method. this code fragment from `XSL2Sample.java` shows how to perform the parse:

   ```
   xmlURL = DemoUtil.createURL(args[1]);
   parser.parse(xmlURL);
   xml = parser.getDocument();
   ```

4. Create a new XSLT stylesheet object. You can pass objects of these classes to the `XSLProcessor.newXSLStylesheet()` method:
   - `java.io.Reader`
   - `java.io.InputStream`
   - `XMLDocument`
   - `java.net.URL`

Copyright © 1999, 2025, Oracle and/or its affiliates.

For example, `XSL2Sample.java` shows how to create a stylesheet object from an `XMLDocument` object:

```
XSLProcessor processor = new XSLProcessor();
processor.setBaseURL(xslURL);
XSLStylesheet xsl = processor.newXSLStylesheet(xsldoc);
```

5. Set the XSLT processor to display any warnings. For example, `XSL2Sample.java` invokes the `showWarnings()` and `setErrorStream()` methods:

```
processor.showWarnings(true);
processor.setErrorStream(System.err);
```

6. Use the `XSLProcessor.processXSL()` method to apply the stylesheet to the input XML data document. Table 14-3 lists some other available `XSLProcessor` methods.

Table 14-3 XSLProcessor Methods

Method	Description
`removeParam()`	Removes parameters.
`resetParams()`	Resets all parameters.
`setParam()`	Sets parameters for the transformation.
`setBaseUrl()`	Sets a base URL for any relative references in the stylesheet.
`setEntityResolver()`	Sets an entity resolver for any relative references in the stylesheet.
`setLocale()`	Sets a locale for error reporting.

This code fragment from `XSL2Sample.java` shows how to apply the stylesheet to the XML document:

```
processor.processXSL(xsl, xml, System.out);
```

7. Process the transformed output. You can transform the results by creating an XML document object, writing to an output stream, or reporting SAX events.

   This code fragment from `XSL2Sample.java` shows how to print the results:

```
processor.processXSL(xsl, xml, System.out);
```

> **See Also**
>
> - *XSL Transformations (XSLT)*
> - *The Extensible Stylesheet Language Family (XSL)*

Related Topics

- XML Parsing for Java
  Extensible Markup Language (XML) parsing for Java is described.

Copyright © 1999, 2025, Oracle and/or its affiliates.

14.3.2 Getting DOM Results from an XSL Transformation

Sample programs show how to obtain the results from an XSL transformation.

The `XSLSample.java` demo program shows how to generate an `oracle.xml.parser.v2.XMLDocumentFragment` object as the result of an XSL transformation. An `XMLDocumentFragment` is a *lightweight* `Document` object that extracts a portion of an XML document tree. The `XMLDocumentFragment` class implements the `org.w3c.dom.DocumentFragment` interface.

The `XSL2Sample.java` demo program shows how to generate a `DocumentFragment` object. The basic steps for transforming XML are the same as those described in Performing Basic XSL Transformation. The only difference is in the arguments passed to the `XSLProcessor.processXSL()` method. This code fragment from `XSL2Sample.java` shows how to create the DOM fragment and then print it to standard output:

```
XMLDocumentFragment result = processor.processXSL(xsl, xml);
result.print(System.out);
```

Table 14-4 lists some `XMLDocumentFragment` methods you can use to manipulate the object.

Table 14-4 XMLDocumentFragment Methods

Method	Description
`getAttributes()`	Gets a `NamedNodeMap` containing the attributes of this node (if it is an `Element`) or null otherwise
`getLocalName()`	Gets the local name for this element
`getNamespaceURI()`	Gets the namespace URI of this element
`getNextSibling()`	Gets the node immediately following the current node
`getNodeName()`	Gets the name of the node
`getNodeType()`	Gets a code that represents the type of the underlying object
`getParentNode()`	Gets the parent of the current node
`getPreviousSibling()`	Gets the node immediately preceding the current node
`reportSAXEvents()`	Reports SAX events from a DOM tree

14.4 Programming with Oracle XSLT Extensions

Topics here include an overview, specifying namespaces for extension functions, using Java methods, using constructor extension functions, and using return value extension functions.

14.4.1 Overview of Oracle XSLT Extensions

The XSLT 1.0 standard defines two kinds of extensions: extension elements and extension functions. XDK provides extension functions for XSLT processing that enable users of the XSLT processor to invoke any Java method from XSL expressions. When using Oracle XSLT extensions, follow these guidelines:

- When you define an XSLT extension in a given programming language, you can use only the XSLT stylesheet with XSLT processors that can invoke this extension. Thus, only the Java version of the processor can invoke extension functions that are defined in Java.

Copyright © 1999, 2025, Oracle and/or its affiliates.

- Use XSLT extensions only if the built-in XSL functions cannot solve a given problem.
- As explained in this section, the namespace of the extension class must start with the proper URL.

These Oracle extension functions are especially useful:

- `<ora:output>`, you can use `<ora:output>` as a top-level element or in an XSL template. If used as a top-level element, it is similar to the `<xsl:output>` extension function, except that it has an additional `name` attribute. When used as a template, it has the additional attributes `use` and `href`. This function is useful for creating multiple outputs from one XSL transformation.
- `<ora:node-set>`, which converts a result tree fragment into a node-set. This function is useful when you want to refer the existing text or intermediate text results in XSL for further transformation.

14.4.2 Specifying Namespaces for XSLT Extension Functions

The Oracle Java extension functions belong to the namespace that corresponds to this Universal Resource Identifier (URI): `http://www.oracle.com/XSL/Transform/java/`. An extension function that belongs to this namespace refers to methods in the Java *`classname`*, so that you can construct URIs in this format: `http://www.oracle.com/XSL/Transform/java/`*`classname`*.

For example, you can use this namespace to invoke `java.lang.String` methods from XSL expressions: `http://www.oracle.com/XSL/Transform/java/java.lang.String`.

> **Note**
>
> When assigning the `xsl` prefix to a namespace, the correct URI is `xmlns:xsl="http://www.w3.org/1999/XSL/Transform"`. Any other URI fails to give correct output.

14.4.3 Using Static and Nonstatic Java Methods in XSLT

If a Java method is a *non*static method of a class then the first parameter is used as the instance on which the method is invoked, and the rest of the parameters are passed to the method. If the extension function is a *static* method, however, then *all* the parameters of the extension function are passed as parameters to the static function.

Example 14-4 shows how to use the `java.lang.Math.ceil()` method in an XSLT stylesheet.

For example, you can create Example 14-4 as stylesheet `ceil.xsl` and then apply it to any well-formed XML document. For example, run the `oraxsl` utility:

```
oraxsl ceil.xsl ceil.xsl ceil.out
```

The output document `ceil.out` has this content:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
13
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

> **Note**
>
> The XSL class loader recognizes only statically added JARs and paths in the `CLASSPATH` and those specified by `wrapper.classpath`. Files added dynamically are not visible to XSLT processor.

Example 14-4 Using a Static Function in an XSLT Stylesheet

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  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>
```

14.4.4 Using Constructor Extension Functions

The extension function `new` creates a new instance of a class and acts as the constructor.

Example 14-5 creates a new `String` object with the value `Hello World`, stores it in the XSL variable `str1`, and then outputs it in uppercase.

For example, you can create this stylesheet as `hello.xsl` and apply it to any well-formed XML document. For example, run the `oraxsl` utility:

```
oraxsl hello.xsl hello.xsl hello.out
```

The output document `hello.out` has this content:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
HELLO WORLD
```

Example 14-5 Using a Constructor in an XSLT Stylesheet

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  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>
```

14.4.5 Using Return Value Extension Functions

The result of an extension function can be of any type, including the five types defined in XSL and the additional simple XML Schema data types defined in XSLT 2.0:

- `NodeSet`
- `Boolean`
- `String`
- `Number`
- `ResultTree`

Copyright © 1999, 2025, Oracle and/or its affiliates.

You can store these data types in variables or pass them to other extension functions. If the result is one of the five types defined in XSL, it can be returned as the result of an XSL expression.

The XSLT Processor supports overloading based on the number of parameters and type. The processor performs implicit type conversion between the five XSL types as defined in XSL. It performs type conversion implicitly among these data types, and also from `NodeSet` to these data types:

- `String`
- `Number`
- `Boolean`
- `ResultTree`

Overloading based on two types that can be implicitly converted to each other is not permitted. This overloading causes an error in XSL because `String` and `Number` can be implicitly converted to each other:

- `overloadme(int i){}`
- `overloadme(String s){}`

Mapping between XSL data types and Java data types is done as follows:

```
String     ->     java.lang.String
Number     ->     int, float, double
Boolean    ->     boolean
NodeSet    ->     NodeList
ResultTree ->     XMLDocumentFragment
```

The stylesheet in Example 14-6 parses the `variable.xml` document, which is located in the directory `$ORACLE_HOME/xdk/demo/java/parser/xslt`, and retrieves the value of the `<title>` child of the `<chapter>` element.

You can create Example 14-6 as `gettitle.xsl` and then run `oraxsl`:

```
oraxsl gettitle.xsl gettitle.xsl variable.out
```

The output document `variable.out` has this content:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
The value of the title element is: Section Tests
```

Example 14-6 gettitle.xsl

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  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 and store it in myparser variable -->
    <xsl:variable name="myparser" select="parser:new()"/>

    <!-- Call an instance method of DOMParser. The first parameter is the object.
     The PI is equivalent to $myparser.parse('file:/my_path/variable.xml'). Note
     that you should replace my_path with the absolute path on your system. -->
    <xsl:value-of select="parser:parse($myparser, 'file:/my_path/variable.xml')"/>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
    <!-- 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,'chapter')">
      The value of the title element is: <xsl:value-of select="docinfo/title" />
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

14.5 Tips and Techniques for Transforming XML

Topics here include using XSLT to merge XML documents and creating an HTML input form based on the columns of a database table.

14.5.1 Merging XML Documents with XSLT

Examples show how to merge XML documents using XSLT.

Merging Documents with appendChild() discusses the DOM technique for merging documents. If the merging operation is simple, then you can also use an XSLT-based approach. For example, you might want to merge the XML documents shown in Example 14-7 and Example 14-8.

Example 14-9 displays a sample stylesheet that merges the two XML documents based on matching the <key/> element values.

Create the XML files in Example 14-7, Example 14-8, and Example 14-9 and run this at the command line:

```
oraxsl msg_w_num.xml msgmerge.xsl msgmerge.xml
```

Example 14-10 shows the output document, which merges the data contained in `msg_w_num.xml` and `msg_w_text.xml`.

This technique is not as efficient for larger files as an equivalent database join of two tables, but it is useful if you have only XML files.

Example 14-7 msg_w_num.xml

```
<messages>
  <msg>
    <key>AAA</key>
    <num>01001</num>
  </msg>
  <msg>
    <key>BBB</key>
    <num>01011</num>
  </msg>
</messages>
```

Example 14-8 msg_w_text.xml

```
<messages>
  <msg>
    <key>AAA</key>
    <text>This is a Message</text>
  </msg>
  <msg>
    <key>BBB</key>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
    <text>This is another Message</text>
  </msg>
</messages>
```

Example 14-9 msgmerge.xsl

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output indent="yes"/>
  <!-- store msg_w_text.xml in doc2 variable -->
  <xsl:variable name="doc2" select="document('msg_w_text.xml')"/>

  <!-- match each node in input xml document, that is, msg_w_num.xml -->
  <xsl:template match="@*|node()">
     <!-- copy the current node to the result tree -->
     <xsl:copy>
        <xsl:apply-templates select="@*|node()"/>
     </xsl:copy>
  </xsl:template>

  <!-- match each <msg> element in msg_w_num.xml -->
  <xsl:template match="msg">
     <xsl:copy>
        <xsl:apply-templates select="@*|node()"/>
        <!-- insert two spaces so indentation is correct in output document -->
        <xsl:text>  </xsl:text>
        <!-- copy <text> node from msg_w_text.xml into result tree -->
        <text><xsl:value-of
               select="$doc2/messages/msg[key=current()/key]/text"/>
        </text>
     </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

Example 14-10 msgmerge.xml

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<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>
```

14.5.2 Creating an HTML Input Form Based on the Columns in a Table

To generate an HTML form for inputting data that uses column names from a database table, you can use the XML SQL Utility (XSU) to get an XML document based on the `user_tab_columns` table and then use XSLT to transform the XML into an HTML form.

1. Use XSU to generate an XML document based on the columns in the table. For example, using the table `hr.employees`, you can run XSU from the command line:

   ```
   java OracleXML getXML -user "hr/password"\
    "SELECT column_name FROM user_tab_columns WHERE table_name = 'EMPLOYEES'"
   ```


Copyright © 1999, 2025, Oracle and/or its affiliates.

2. Save the XSU output as an XML file called `emp_columns.xml`. The XML looks like this, with one `<ROW>` element corresponding to each column in the table (some `<ROW>` elements have been removed to conserve space):

```
<?xml version = '1.0'?><ROWSET>
   <ROW num="1">
      <COLUMN_NAME>EMPLOYEE_ID</COLUMN_NAME>
   </ROW>
   <ROW num="2">
      <COLUMN_NAME>FIRST_NAME</COLUMN_NAME>
   </ROW>
   <!-- rows 3 through 10 -->
   <ROW num="11">
      <COLUMN_NAME>DEPARTMENT_ID</COLUMN_NAME>
   </ROW>
</ROWSET>
```

3. Create an XSLT stylesheet to transform the XML into HTML. For example, create the `columns.xsl` stylesheet:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <HTML>
      <xsl:apply-templates select="@*|node()"/>
    </HTML>
  </xsl:template>
  <xsl:template match="ROW">
    <xsl:value-of select="COLUMN_NAME"/>
    <xsl:text> </xsl:text>
    <INPUT NAME="{COLUMN_NAME}"/>
    <BR/>
  </xsl:template>
</xsl:stylesheet>
```

4. Run the `oraxsl` utility to generate the HTML form. For example:

```
oraxsl emp_columns.xml columns.xsl emp_form.htm
```

5. Review the output HTML form, which has contents similar to these:

```
<HTML>
   EMPLOYEE_ID <INPUT NAME="EMPLOYEE_ID"><BR>
   FIRST_NAME <INPUT NAME="FIRST_NAME"><BR>
   LAST_NAME <INPUT NAME="LAST_NAME"><BR>
   EMAIL <INPUT NAME="EMAIL"><BR>
   PHONE_NUMBER <INPUT NAME="PHONE_NUMBER"><BR>
   HIRE_DATE <INPUT NAME="HIRE_DATE"><BR>
   JOB_ID <INPUT NAME="JOB_ID"><BR>
   SALARY <INPUT NAME="SALARY"><BR>
   COMMISSION_PCT <INPUT NAME="COMMISSION_PCT"><BR>
   MANAGER_ID <INPUT NAME="MANAGER_ID"><BR>
   DEPARTMENT_ID <INPUT NAME="DEPARTMENT_ID"><BR>
</HTML>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

15

Using the XQuery Processor for Java

An explanation is given of how to use the Oracle XML Developer's Kit (XDK) XQuery processor for Java.

15.1 Introduction to the XQuery Processor for Java

XDK provides a standalone XQuery processor for use by Java applications. XQuery is the World Wide Web Consortium (W3C) standard query language for Extensible Markup Language (XML). Using XQuery to process XML within a Java application can improve developer productivity and application performance.

Applications written with XQuery often require less code, run faster, and use less memory than applications written fully in Java.

JSR 225: XQuery API for Java (XQJ) defines how queries can be executed from a Java application. To use XQJ, your application must run with Java version 1.6. In addition, these JAR files are required:

- `jlib/oxquery.jar`
- `jlib/xqjapi.jar`
- `jlib/orai18n-mapping.jar`
- `lib/xmlparserv2.jar`
- `xdk/jlib/apache-xmlbeans.jar`

The directory paths for these Java Archive (JAR) files are relative to the `ORACLE_HOME` directory of your Oracle Database installation.

Example 15-1 shows how to execute a simple "Hello World" query using XQuery API for Java (XQJ). Because the XQuery processor runs directly in the Java Virtual Machine (JVM), you need no database or server to run this example. The example prints the output `<hello-world>2</hello-world>`.

This chapter describes the features and extensions that are specific to the Oracle implementation of XQuery. General information about XQuery and XQJ is documented outside of this document.

See Also

- *Oracle AI Database XML Java API Reference*, `XQuery Packages`, for the related API documentation
- *JSR-000225 XQuery API for Java*
- *XQuery 3.0: An XML Query Language*

Copyright © 1999, 2025, Oracle and/or its affiliates.

> **Note**
>
> Oracle also implements XQuery and XQJ as part of Oracle XML DB. See Using XQuery API for Java to Access Oracle XML DB for details about Oracle XML DB.

Example 15-1 Simple Query Using XQJ

```
import javax.xml.xquery.XQConnection;
import javax.xml.xquery.XQException;
import javax.xml.xquery.XQPreparedExpression;
import javax.xml.xquery.XQSequence;

import oracle.xml.xquery.OXQDataSource;

public class HelloWorld {

    public static void main(String[] args) throws XQException {
        OXQDataSource ds = new OXQDataSource();
        XQConnection con = ds.getConnection();
        String query = "<hello-world>{1 + 1}</hello-world>";
        XQPreparedExpression expr = con.prepareExpression(query);
        XQSequence result = expr.executeQuery();

        // prints "<hello-world>2</hello-world>"
        System.out.println(result.getSequenceAsString(null));

        result.close();
        expr.close();
        con.close();
    }

}
```

15.2 XQJ Entity Resolution

XDK extends XQJ with an entity resolver framework for controlling how documents, schemas, modules, collations, and external functions are obtained during query processing. The examples in this section show how to use an entity resolver for several types of entities.

See the class `oracle.xml.xquery.OXQEntity` in *Oracle AI Database XML Java API Reference* for a complete list of the types of entities that the query processor can request.

15.2.1 Resolution of Documents for fn:doc

The example in this section shows how you can use an entity resolver to determine which document is returned by XQuery function `fn:doc`.

Example 15-2 displays the contents of `books.xml`.

Example 15-3 displays the contents of `books.xq`.

Example 15-4 shows how to execute the query `books.xq` with a custom entity resolver.

The instance of `MyEntityResolver` is passed to the XQuery processor by setting it on the connection. The XQuery processor invokes the entity resolver during query processing to get the document to be returned by the `fn:doc` function.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Example 15-2 books.xml

```
<books>
  <book>
    <title>A Game of Thrones</title>
    <author><first>George</first><last>Martin</last></author>
    <price>10.99</price>
  </book>
  <book>
    <title>The Pillars of the Earth</title>
    <author><first>Ken</first><last>Follett</last></author>
    <price>7.99</price>
  </book>
</books>
```

Example 15-3 books.xq

```
for $book in fn:doc('books.xml')/books/book
where xs:decimal($book/price) gt 10.00
return
  $book/title
```

Example 15-4 Executing a Query with a Custom Entity Resolver

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.net.URI;

import javax.xml.xquery.XQConnection;
import javax.xml.xquery.XQException;
import javax.xml.xquery.XQPreparedExpression;
import javax.xml.xquery.XQSequence;
import javax.xml.xquery.XQStaticContext;

import oracle.xml.xquery.OXQConnection;
import oracle.xml.xquery.OXQDataSource;
import oracle.xml.xquery.OXQEntity;
import oracle.xml.xquery.OXQEntityKind;
import oracle.xml.xquery.OXQEntityLocator;
import oracle.xml.xquery.OXQEntityResolver;
import oracle.xml.xquery.OXQEntityResolverRequestOptions;
import oracle.xml.xquery.OXQView;

public class ResolveDocument {

    private static class MyEntityResolver extends OXQEntityResolver {
        @Override
        public OXQEntity resolveEntity(OXQEntityKind kind, OXQEntityLocator locator,
                OXQEntityResolverRequestOptions options) throws IOException {
            if (kind == OXQEntityKind.DOCUMENT) {
                URI systemId = locator.getSystemIdAsURI();
                if ("file".equals(systemId.getScheme())) {
                    File file = new File(systemId);
                    FileInputStream input = new FileInputStream(file);
                    OXQEntity result = new OXQEntity(input);
                    result.enlistCloseable(input);
                    return result;
                }
            }
            return null;
        }
    }

    public static void main(String[] args) throws XQException, IOException {
        OXQDataSource ds = new OXQDataSource();
        XQConnection con = ds.getConnection();
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
        // OXQView is used to access Oracle extensions on XQJ objects.
        OXQConnection ocon = OXQView.getConnection(con);
        ocon.setEntityResolver(new MyEntityResolver());

        File query = new File("books.xq");

        // Relative URIs are resolved against the base URI before invoking the entity resolver.
        // The relative URI 'books.xml' used in the query will be resolved against this URI.
        XQStaticContext ctx = con.getStaticContext();
        ctx.setBaseURI(query.toURI().toString());

        FileInputStream queryInput = new FileInputStream(query);
        XQPreparedExpression expr = con.prepareExpression(queryInput, ctx);
        queryInput.close();
        XQSequence result = expr.executeQuery();

        // Prints "<title>A Game of Thrones</title>"
        System.out.println(result.getSequenceAsString(null));

        result.close();
        expr.close();
        con.close();
    }
}
```

The example generates this output:

```
<title>A Game of Thrones</title>
```

15.2.2 Resolution of External XQuery Functions

You can use an entity resolver to define the implementation of an XQuery external function.

For each external XQuery function that is declared in a query, the entity resolver is called with the entity kind `oracle.xml.xquery.OXQEntityKind.EXTERNAL_FUNCTION`. The `oracle.xml.xquery.OXQEntityLocator` instance that is passed in the call to the entity resolver provides the name of the XQuery function and its argument types. The entity resolver can then return any class that extends `oracle.xml.xquery.OXQFunctionEvaluator` and has a public constructor. The XQuery processor then instantiates the returned class. When the XQuery external function call is evaluated, method `evaluate()` is invoked.

Example 15-5 displays an XQuery query that is the content of file `trim.xq`.

External XQuery function `util:trim` removes white space from the beginning and end of a string value. This function is implemented in Java and called within the query.

Example 15-6 uses `trim.xq` and shows how to define the implementation of an external XQuery function. The entity resolver in this example returns a class that extends `OXQFunctionEvaluator`.

In some cases it is more convenient to return a Java static method instead of a class. When a static method is returned, the query processor automatically maps the method arguments and the return value to the XQuery data model, as defined by the XQJ specification.

Example 15-7 also runs `trim.xq`, but in this case the external function is bound to a Java static method.

Example 15-5 trim.xq

```
declare namespace util = "http://example.com/util";

declare function util:trim($arg as xs:string) as xs:string external;

(: a string with surrounding white space :)
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
declare variable $input := "   John Doe    ";

<result>{util:trim($input)}</result>
```

Example 15-6 Defining the Implementation of an External XQuery Function

```
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Collections;

import javax.xml.namespace.QName;
import javax.xml.xquery.XQConnection;
import javax.xml.xquery.XQException;
import javax.xml.xquery.XQPreparedExpression;
import javax.xml.xquery.XQSequence;

import oracle.xml.xquery.OXQConnection;
import oracle.xml.xquery.OXQDataSource;
import oracle.xml.xquery.OXQEntity;
import oracle.xml.xquery.OXQEntityKind;
import oracle.xml.xquery.OXQEntityLocator;
import oracle.xml.xquery.OXQEntityResolver;
import oracle.xml.xquery.OXQEntityResolverRequestOptions;
import oracle.xml.xquery.OXQFunctionContext;
import oracle.xml.xquery.OXQFunctionEvaluator;
import oracle.xml.xquery.OXQFunctionMetaData;
import oracle.xml.xquery.OXQView;

public class ResolveExternalFunction {

    public static class TrimFunction extends OXQFunctionEvaluator {
        @Override
        public XQSequence evaluate(OXQFunctionContext context, XQSequence[] params) throws XQException {
            XQConnection con = context.getConnection();
            XQSequence arg = params[0];
            String value = arg.getSequenceAsString(null);
            String trimmed = value.trim();
            return con.createSequence(Collections.singleton(trimmed).iterator());
        }
    }

    private static class MyEntityResolver extends OXQEntityResolver {
        @Override
        public OXQEntity resolveEntity(OXQEntityKind kind, OXQEntityLocator locator,
                OXQEntityResolverRequestOptions options) throws XQException, IOException {
            if (kind == OXQEntityKind.EXTERNAL_FUNCTION) {
                OXQFunctionMetaData metaData = (OXQFunctionMetaData)locator.getExtension();
                QName name = metaData.getName();
                int arity = metaData.getParameterTypes().length;
                if ("http://example.com/util".equals(name.getNamespaceURI()) &&
                    "trim".equals(name.getLocalPart()) && arity == 1) {
                    return new OXQEntity(TrimFunction.class);
                }
            }
            return null;
        }
    }

    public static void main(String[] args) throws IOException, XQException {
        OXQDataSource ds = new OXQDataSource();
        XQConnection con = ds.getConnection();
        OXQConnection ocon = OXQView.getConnection(con);
        ocon.setEntityResolver(new MyEntityResolver());

        FileInputStream query = new FileInputStream("trim.xq");
        XQPreparedExpression expr = con.prepareExpression(query);
        query.close();

        XQSequence result = expr.executeQuery();
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
        System.out.println(result.getSequenceAsString(null));

        result.close();
        expr.close();
        con.close();
    }
}
```

The example prints this output: `<result>John Doe</result>`.

Example 15-7 Binding an External Function to a Java Static Method

```
import java.io.FileInputStream;
import java.io.IOException;
import java.lang.reflect.Method;

import javax.xml.namespace.QName;
import javax.xml.xquery.XQConnection;
import javax.xml.xquery.XQException;
import javax.xml.xquery.XQPreparedExpression;
import javax.xml.xquery.XQSequence;

import oracle.xml.xquery.OXQConnection;
import oracle.xml.xquery.OXQDataSource;
import oracle.xml.xquery.OXQEntity;
import oracle.xml.xquery.OXQEntityKind;
import oracle.xml.xquery.OXQEntityLocator;
import oracle.xml.xquery.OXQEntityResolver;
import oracle.xml.xquery.OXQEntityResolverRequestOptions;
import oracle.xml.xquery.OXQFunctionMetaData;
import oracle.xml.xquery.OXQView;

public class ResolveExternalFunction2 {

    public static String trim(String value) {
        return value.trim();
    }

    private static class MyEntityResolver extends OXQEntityResolver {
        @Override
        public OXQEntity resolveEntity(OXQEntityKind kind, OXQEntityLocator locator,
                OXQEntityResolverRequestOptions options) throws XQException, IOException {
            if (kind == OXQEntityKind.EXTERNAL_FUNCTION) {
                OXQFunctionMetaData metaData = (OXQFunctionMetaData)locator.getExtension();
                QName name = metaData.getName();
                int arity = metaData.getParameterTypes().length;
                if ("http://example.com/util".equals(name.getNamespaceURI()) &&
                    "trim".equals(name.getLocalPart()) && arity == 1) {
                    Method staticMethod = null;
                    try {
                        staticMethod = ResolveExternalFunction2.class.getMethod("trim", String.class);
                    } catch (NoSuchMethodException e) {
                        throw new IllegalStateException(e);
                    }
                    return new OXQEntity(staticMethod);
                }
            }
            return null;
        }
    }

    public static void main(String[] args) throws IOException, XQException {
        OXQDataSource ds = new OXQDataSource();
        XQConnection con = ds.getConnection();
        OXQConnection ocon = OXQView.getConnection(con);
        ocon.setEntityResolver(new MyEntityResolver());

        FileInputStream query = new FileInputStream("trim.xq");
        XQPreparedExpression expr = con.prepareExpression(query);
        query.close();
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
        XQSequence result = expr.executeQuery();

        // Prints "<result>John Doe</result>"
        System.out.println(result.getSequenceAsString(null));

        result.close();
        expr.close();
        con.close();
    }
}
```

The example prints the same output as for Example 15-6: `<result>John Doe</result>`.

15.2.3 Resolution of Imported XQuery Modules

The entity resolver can locate the XQuery modules imported by an XQuery library module.

An XQuery library module provides functions and variables that can be imported by other modules. For each imported module, the entity resolver is called with the entity kind `oracle.xml.xquery.OXQEntityKind.MODULE`. Using the `oracle.xml.xquery.OXQEntityLocator` instance, you can invoke the `getSystemId()` method to get the location of the module being imported. If no location is provided in the module import, you can invoke the method `getNamespace()` to get the target namespace of the module. The entity resolver can then return the corresponding library module.

The example in this section shows how you can use an entity resolver to control the resolution of XQuery library modules.

Example 15-8 displays the contents of `math.xq`.

Example 15-9 displays the contents of `main.xq`.

Example 15-10 shows how to execute a query that imports a library module.

The query `main.xq` imports the library module `math.xq`, and then invokes the function `math:circumference` to compute the circumference of a circle.

Example 15-8 math.xq

```
module namespace math = "http://example.com/math";

declare variable $math:pi as xs:decimal := 3.14159265;

declare function math:circumference($diameter as xs:decimal) {
   $math:pi * $diameter
};
```

Example 15-9 main.xq

```
import module namespace math = "http://example.com/math" at "math.xq";

math:circumference(6.54)
```

Example 15-10 Executing a Query that Imports a Library Module

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.net.URI;

import javax.xml.xquery.XQConnection;
import javax.xml.xquery.XQException;
import javax.xml.xquery.XQPreparedExpression;
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
import javax.xml.xquery.XQSequence;
import javax.xml.xquery.XQStaticContext;

import oracle.xml.xquery.OXQConnection;
import oracle.xml.xquery.OXQDataSource;
import oracle.xml.xquery.OXQEntity;
import oracle.xml.xquery.OXQEntityKind;
import oracle.xml.xquery.OXQEntityLocator;
import oracle.xml.xquery.OXQEntityResolver;
import oracle.xml.xquery.OXQEntityResolverRequestOptions;
import oracle.xml.xquery.OXQView;

public class ResolveLibraryModule {

    private static class MyEntityResolver extends OXQEntityResolver {
        @Override
        public OXQEntity resolveEntity(OXQEntityKind kind, OXQEntityLocator locator,
                OXQEntityResolverRequestOptions options) throws IOException {
            if (kind == OXQEntityKind.MODULE) {
                URI systemId = locator.getSystemIdAsURI();
                if (systemId != null && "file".equals(systemId.getScheme())) {
                    File file = new File(systemId);
                    FileInputStream input = new FileInputStream(file);
                    OXQEntity result = new OXQEntity(input);
                    result.enlistCloseable(input);
                    return result;
                }
            }
            return null;
        }
    }

    public static void main(String[] args) throws XQException, IOException {
        OXQDataSource ds = new OXQDataSource();
        XQConnection con = ds.getConnection();

        // OXQView is used to access Oracle extensions on XQJ objects.
        OXQConnection ocon = OXQView.getConnection(con);
        ocon.setEntityResolver(new MyEntityResolver());

        File query = new File("main.xq");

        // Relative URIs are resolved against the base URI before invoking the entity resolver.
        // The relative URI 'math.xq' used in the query will be resolved against this URI.
        XQStaticContext ctx = con.getStaticContext();
        ctx.setBaseURI(query.toURI().toString());

        FileInputStream queryInput = new FileInputStream(query);
        XQPreparedExpression expr = con.prepareExpression(queryInput, ctx);
        queryInput.close();

        XQSequence result = expr.executeQuery();

        // Prints the result: "20.546015931"
        System.out.println(result.getSequenceAsString(null));

        result.close();
        expr.close();
        con.close();
    }
}
```

The example generates this output:

```
20.546015931
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

15.2.4 Resolution of XML Schemas Imported by an XQuery Query

You can use an entity resolver to control which XML schema is imported by an XQuery query.

An XQuery schema import imports type definitions and declarations of elements and attributes from an XML schema. You can use imported declarations and definitions in a query to validate and test data instances.

Example 15-11 displays the contents of XML schema file `size.xsd`.

Example 15-12 displays the contents of XQuery file `size.xq`.

Example 15-13 shows how to execute a query that imports a schema.

`size.xq` uses the type `shirt-size` defined in `size.xsd` to test a list of values.

Example 15-11 size.xsd

```
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://example.com/size">

  <xs:simpleType name="shirt-size">
    <xs:restriction base="xs:string">
      <xs:enumeration value="XS"/>
      <xs:enumeration value="S"/>
      <xs:enumeration value="M"/>
      <xs:enumeration value="L"/>
      <xs:enumeration value="XL"/>
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

Example 15-12 size.xq

```
import schema namespace ns = "http://example.com/size" at "size.xsd";

for $size in ("S", "big", "XL", 42)
return
  if ($size castable as ns:shirt-size) then
    ns:shirt-size($size)
  else
    concat("INVALID:", $size)
```

Example 15-13 Executing an XQuery Query that Imports an XML Schema

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.net.URI;

import javax.xml.xquery.XQConnection;
import javax.xml.xquery.XQException;
import javax.xml.xquery.XQPreparedExpression;
import javax.xml.xquery.XQSequence;
import javax.xml.xquery.XQStaticContext;

import oracle.xml.xquery.OXQConnection;
import oracle.xml.xquery.OXQDataSource;
import oracle.xml.xquery.OXQEntity;
import oracle.xml.xquery.OXQEntityKind;
import oracle.xml.xquery.OXQEntityLocator;
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
import oracle.xml.xquery.OXQEntityResolver;
import oracle.xml.xquery.OXQEntityResolverRequestOptions;
import oracle.xml.xquery.OXQView;

public class ResolveSchema {

    private static class MyEntityResolver extends OXQEntityResolver {
        @Override
        public OXQEntity resolveEntity(OXQEntityKind kind, OXQEntityLocator locator,
                OXQEntityResolverRequestOptions options) throws IOException {
            if (kind == OXQEntityKind.SCHEMA) {
                URI systemId = locator.getSystemIdAsURI();
                if (systemId != null && "file".equals(systemId.getScheme())) {
                    File file = new File(systemId);
                    FileInputStream input = new FileInputStream(file);
                    OXQEntity result = new OXQEntity(input);
                    result.enlistCloseable(input);
                    return result;
                }
            }
            return null;
        }
    }

   public static void main(String[] args) throws XQException, IOException {
        OXQDataSource ds = new OXQDataSource();
        XQConnection con = ds.getConnection();

        // OXQView is used to access Oracle extensions on XQJ objects.
        OXQConnection ocon = OXQView.getConnection(con);
        ocon.setEntityResolver(new MyEntityResolver());

        File query = new File("size.xq");

        // Relative URIs are resolved against the base URI before invoking the entity resolver.
        // The relative URI 'math.xq' used in the query will be resolved against this URI.
        XQStaticContext ctx = con.getStaticContext();
        ctx.setBaseURI(query.toURI().toString());

        FileInputStream queryInput = new FileInputStream(query);
        XQPreparedExpression expr = con.prepareExpression(queryInput, ctx);
        queryInput.close();

        XQSequence result = expr.executeQuery();

        // Prints "S INVALID:big XL INVALID:42"
        System.out.println(result.getSequenceAsString(null));

        result.close();
        expr.close();
        con.close();
    }
}
```

The example prints this output: `S INVALID:big XL INVALID:42`.

15.2.5 Prefabricated Entity Resolvers for XQuery

XDK includes several implementations of `OXQEntityResolver` that you can use for common tasks such as file system and HTTP resolution. This can sometimes save you needing to implement your own entity resolver.

Example 15-14 shows how you can run the query in Example 15-3 using a prefabricated file resolver.

An instance of the factory `oracle.xml.xquery.OXQFileResolverFactory` is created from the connection. Then, this factory is used to create an entity resolver that resolves schemas,

Copyright © 1999, 2025, Oracle and/or its affiliates.

modules, and documents against the file system. By contrast with this example, Example 15-4 uses the custom entity resolver `MyEntityResolver` to resolve only documents against the file system.

XDK provides these entity resolver factories:

- `oracle.xml.xquery.OXQFileResolverFactory`: Creates an entity resolver that resolves 'file:' URIs for schema, module, and document locations.
- `oracle.xml.xquery.OXQHttpResolverFactory`: Creates an entity resolver that resolves 'http:' URIs for schema, module, and document locations.
- `oracle.xml.xquery.OXQCompositeResolverFactory`: Creates an entity resolver that delegates requests to other entity resolvers. For any kind of request, the resolver returns the first nonnull result it receives from one of the delegate resolvers.
- `oracle.xml.xquery.OXQJavaResolverFactory`: Creates an entity resolver that resolves external functions and modules to Java static methods or classes.

> **See Also**
>
> *Oracle AI Database XML Java API Reference*, package `oracle.xml.xquery`, for API information about these factory interfaces

Example 15-14 Executing a Query with a Prefabricated File Resolver

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

import javax.xml.xquery.XQConnection;
import javax.xml.xquery.XQException;
import javax.xml.xquery.XQPreparedExpression;
import javax.xml.xquery.XQSequence;
import javax.xml.xquery.XQStaticContext;

import oracle.xml.xquery.OXQConnection;
import oracle.xml.xquery.OXQDataSource;
import oracle.xml.xquery.OXQFileResolverFactory;
import oracle.xml.xquery.OXQView;

public class ResolverFactory {
    public static void main(String[] args) throws XQException, IOException {
        OXQDataSource ds = new OXQDataSource();
        XQConnection con = ds.getConnection();

        // OXQView is used to access Oracle extensions on XQJ objects.
        OXQConnection ocon = OXQView.getConnection(con);
        OXQFileResolverFactory factory = ocon.createEntityResolverFactory(OXQFileResolverFactory.class);
        ocon.setEntityResolver(factory.createResolver());

        File query = new File("books.xq");

        // Relative URIs are resolved against the base URI before invoking the entity resolver.
        // The relative URI 'books.xml' used in the query will be resolved against this URI.
        XQStaticContext ctx = con.getStaticContext();
        ctx.setBaseURI(query.toURI().toString());

        FileInputStream queryInput = new FileInputStream(query);
        XQPreparedExpression expr = con.prepareExpression(queryInput, ctx);
        queryInput.close();

        XQSequence result = expr.executeQuery();
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
        // Prints "<title>A Game of Thrones</title>"
        System.out.println(result.getSequenceAsString(null));

        result.close();
        expr.close();
        con.close();
    }
}
```

The example prints this output: `<title>A Game of Thrones</title>`.

15.2.6 Resolution of Other Types of Entity

You can use the XQJ entity resolver to obtain other entities, besides documents, schemas, modules, and external functions.

Table 15-1 describes these other entities.

Table 15-1 Descriptions of Various Types of Entity

OXQEntityKind	Description
`TEXT`	Result of `fn:unparsed-text` and `fn:unparsed-text-lines`.
`ENVIRONMENT_VARIABLE`	Result of `fn:available-environment-variables` and `fn:environment-variable`.
`DOCUMENT_TYPE`	Static type for function `fn:doc`.
`COLLECTION`	Documents returned by `fn:collection`.
`URI_COLLECTION`	URIs returned by `fn:uri-collection`.
`XML_PARSER_FACTORY`	StAX implementation used for parsing XML data.
`XML_ENTITY`	Used when a StAX parser needs to resolve an external XML resource.
`UPD_PUT`	Controls the behavior of `fn:put`.
`COLLATION`	Controls the behavior of collation URIs.

For details on how these entity types are used, see class `oracle.xml.xquery.OXQEntity` in *Oracle AI Database XML Java API Reference*.

15.3 XQuery Output Declarations

XQuery 3.0 defines output declarations, which you can use to set the values of serialization parameters from within a query.

An output declaration is an option declaration in namespace `http://www.w3.org/2010/xslt-xquery-serialization`. You use it in a query to declare and set serialization parameters on the static context.

By default, these static context serialization parameters are ignored, but they can be accessed using XQJ methods `OXQPreparedExpression#getExpressionStaticContext()` and `OXQStaticContext#getSerializationParameters()`.

Example 15-15 shows how to access the values of option declarations.

You can also use an option declaration when serializing the result of a query. Example 15-16 is similar to Example 15-15, but it uses the static context serialization parameters when serializing the result of the query.

Copyright © 1999, 2025, Oracle and/or its affiliates.

> **Note**
>
> The URI value of option `output:parameter-document` is resolved to a document using entity resolver `OXQEntityResolver` and entity kind `OXQEntityKind#DOCUMENT` — see Resolution of Other Types of Entity.

Example 15-15 Accessing the Values of Option Declarations

```
import javax.xml.xquery.XQConnection;
import javax.xml.xquery.XQException;
import javax.xml.xquery.XQPreparedExpression;
import javax.xml.xquery.XQStaticContext;

import oracle.xml.xquery.OXQDataSource;
import oracle.xml.xquery.OXQPreparedExpression;
import oracle.xml.xquery.OXQSerializationParameters;
import oracle.xml.xquery.OXQStaticContext;
import oracle.xml.xquery.OXQView;

public class OptionDeclarations1 {

    public static void main(String[] args) throws XQException {
        OXQDataSource ds = new OXQDataSource();
        XQConnection con = ds.getConnection();
        String query =
            "declare option output:indent 'yes'; \n" +
            "declare option output:encoding 'UTF-16'; \n" +
            "<person><first>John</first><last>Doe</last></person>";

        XQPreparedExpression expr = con.prepareExpression(query);

        OXQPreparedExpression oexpr = OXQView.getPreparedExpression(expr);
        XQStaticContext ctx = oexpr.getExpressionStaticContext();
        OXQStaticContext octx = OXQView.getStaticContext(ctx);
        OXQSerializationParameters params = octx.getSerializationParameters();

        System.out.println("indent=" + params.isIndent());
        System.out.println("encoding=" + params.getEncoding());

        expr.close();
        con.close();
    }
}
```

This produces the following output:

```
  indent=true
  encoding=UTF-16
```

Example 15-16 Using Option Declarations When Serializing a Query Result

```
package oracle.xml.xquery.examples.published;
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
import javax.xml.xquery.XQConnection;
import javax.xml.xquery.XQException;
import javax.xml.xquery.XQPreparedExpression;
import javax.xml.xquery.XQResultSequence;
import javax.xml.xquery.XQStaticContext;

import oracle.xml.xquery.OXQDataSource;
import oracle.xml.xquery.OXQPreparedExpression;
import oracle.xml.xquery.OXQSerializationParameters;
import oracle.xml.xquery.OXQStaticContext;
import oracle.xml.xquery.OXQView;

public class OptionDeclarations2 {

    public static void main(String[] args) throws XQException {
        OXQDataSource ds = new OXQDataSource();
        XQConnection con = ds.getConnection();
        String query =
            "declare option output:indent 'yes'; \n" +
            "declare option output:omit-xml-declaration 'no'; \n" +
            "<person><first>John</first><last>Doe</last></person>";

        XQPreparedExpression expr = con.prepareExpression(query);

        OXQPreparedExpression oexpr = OXQView.getPreparedExpression(expr);
        XQStaticContext ctx = oexpr.getExpressionStaticContext();
        OXQStaticContext octx = OXQView.getStaticContext(ctx);
        OXQSerializationParameters params = octx.getSerializationParameters();

        XQResultSequence result = expr.executeQuery();
        result.writeSequence(System.out, params.createProperties());
        result.close();

        expr.close();
        con.close();
    }
}
```

This produces the following output:

```
<?xml version="1.0" encoding="UTF-8"?>
<person>
    <first>John</first>
    <last>Doe</last>
</person>
```

15.4 Improving Application Performance and Scalability with XQuery

The XDK XQuery processor provides several features for improving the performance and scalability of your application.

Copyright © 1999, 2025, Oracle and/or its affiliates.

15.4.1 Streaming Query Evaluation

The XDK XQuery processor for Java supports streaming evaluation for many types of queries. Streaming evaluation requires a small amount of main memory, even when the input XML is very large.

To facilitate streaming evaluation, the following actions are recommended:

- Set the binding mode on the static context to deferred mode (see the method `javax.xml.xquery.XQStaticContext.setBindingMode(int)` in *Oracle AI Database XML Java API Reference*). If the binding mode is not deferred, the input XML is fully materialized when it is bound.
- Provide the input XML as an instance of `java.io.InputStream`, `java.io.Reader`, or `javax.xml.stream.XMLStreamReader`. Input XML is provided to the query processor by binding it to the expression, or by returning it from an entity resolver.
- Ensure that the `javax.xml.xquery.XQSequence` instance is consumed in a way that does not require materialization:
  - The string serialization methods `getSequenceAsString(...)` and `getItemAsString(...)` produce data as a string that is held in memory. Instead, use the `writeSequence(...)` or the `writeItem(...)` method to serialize the sequence.
  - The `getNode()` method builds a Document Object Model (DOM) node that is held in memory. Instead, consider using the `getSequenceAsStream()` or the `getItemAsStream()` method to get a Streaming API for XML (StAX) stream.
  - The `getItem()` method copies and materializes the current item in memory. Instead, use methods directly on the `java.xml.xquery.XQSequence` instance to access the current item (see the interface `javax.xml.xquery.XQItemAccessor` in *Oracle AI Database XML Java API Reference*).

The code shown in this section invokes a query using XQJ in a way that does not prevent streaming evaluation.

Example 15-17 displays the contents of `books2.xq`.

Example 15-18 sets up the query to enable streaming evaluation. The example writes this output to file `results.xml`: `<title>A Game of Thrones</title>`.

The binding mode is set to the value `BINDING_MODE_DEFERRED` to avoid materializing `books.xml` when it is bound to the prepared expression. Likewise, the result is written to an output stream, and it is not materialized.

To simplify the example, the input file `books.xml` is small. Even if this file contained millions of books, evaluating the query would require only a small maximum heap size because only one book element is held in memory at one time. In contrast with the query `books.xq`, shown in Example 15-3, the query `books2.xq` does not require you to define an entity resolver. Both examples (`books.xq` and `books2.xq`) are streamable.

Example 15-17 books2.xq

```
declare variable $doc external;

for $book in $doc/books/book
where xs:decimal($book/price) gt 10.00
return
  $book/title
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

Example 15-18 Facilitating Streaming Evaluation

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

import javax.xml.namespace.QName;
import javax.xml.xquery.XQConnection;
import javax.xml.xquery.XQConstants;
import javax.xml.xquery.XQException;
import javax.xml.xquery.XQPreparedExpression;
import javax.xml.xquery.XQSequence;
import javax.xml.xquery.XQStaticContext;

import oracle.xml.xquery.OXQDataSource;

public class Streaming {

    public static void main(String[] args) throws XQException, IOException {
        OXQDataSource ds = new OXQDataSource();
        XQConnection con = ds.getConnection();

        XQStaticContext ctx = con.getStaticContext();
        ctx.setBindingMode(XQConstants.BINDING_MODE_DEFERRED);
        con.setStaticContext(ctx);

        FileInputStream input = new FileInputStream("books.xml");
        FileInputStream query = new FileInputStream("books2.xq");
        FileOutputStream output = new FileOutputStream("result.xml");

        XQPreparedExpression expr = con.prepareExpression(query);
        query.close();
        expr.bindDocument(new QName("doc"), input, null, null);

        XQSequence result = expr.executeQuery();

        // Writes "<title>A Game of Thrones</title>" to file results.xml
        result.writeSequence(output, null);

        result.close();
        input.close();
        output.close();
        expr.close();
        con.close();
    }
}
```

15.4.2 External Storage

Depending on the query, the processor might have to store part of the input XML in main memory during query evaluation.

For example, this scenario can occur in cases such as these:

- A sequence is sorted.
- The value bound to a variable is used multiple times.
- A path expression uses a reverse-axis step.

To reduce memory usage in such cases, you can configure the XQuery processor to use external storage for materializing XML, rather than main memory. To enable the use of external storage, set the data source property `OXQConstants.USE_EXTERNAL_STORAGE` to `true`, and set an `oracle.xml.scalable.PageManager` instance on the dynamic context.

Copyright © 1999, 2025, Oracle and/or its affiliates.

> **Note**
>
> Using external storage can significantly reduce the amount of main memory that is consumed during query processing. However, it can also reduce performance.

Example 15-19 shows how to enable the XQuery processor to use disk-based storage rather than main memory when XML is materialized. The example writes this output to file `results.xml`: `<title>A Game of Thrones</title>`.

Example 15-19 Configuring the XQuery Processor to Use External Storage

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

import javax.xml.namespace.QName;
import javax.xml.xquery.XQConnection;
import javax.xml.xquery.XQConstants;
import javax.xml.xquery.XQException;
import javax.xml.xquery.XQPreparedExpression;
import javax.xml.xquery.XQSequence;
import javax.xml.xquery.XQStaticContext;

import oracle.xml.scalable.FilePageManager;
import oracle.xml.xquery.OXQDataSource;
import oracle.xml.xquery.OXQPreparedExpression;
import oracle.xml.xquery.OXQView;

public class ExternalStorage {

    public static void main(String[] args) throws XQException, IOException {
        OXQDataSource ds = new OXQDataSource();
        ds.setProperty(OXQDataSource.USE_EXTERNAL_STORAGE, "true");

        XQConnection con = ds.getConnection();
        XQStaticContext ctx = con.getStaticContext();
        ctx.setBindingMode(XQConstants.BINDING_MODE_DEFERRED);
        con.setStaticContext(ctx);

        FileInputStream input = new FileInputStream("books.xml");
        FileInputStream query = new FileInputStream("books2.xq");
        FileOutputStream output = new FileOutputStream("results.xml");

        XQPreparedExpression expr = con.prepareExpression(query);
        query.close();
        expr.bindDocument(new QName("doc"), input, null, null);

        // Set a page manager that will be used by the XQuery processor if XML needs to be materialized
        OXQPreparedExpression oexpr = OXQView.getPreparedExpression(expr);
        File temporaryFile = File.createTempFile("books", ".pagefile");
        temporaryFile.deleteOnExit();
        oexpr.setPageManager(new FilePageManager(temporaryFile.getAbsolutePath()));

        XQSequence result = expr.executeQuery();

        // Writes to file results.xml: "<title>A Game of Thrones</title>"
        result.writeSequence(output, null);

        result.close();
        input.close();
        output.close();
        expr.close();
        con.close();
    }
}
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

15.4.3 Thread Safety for XQJ

The Oracle implementation of XQJ is not thread-safe. For example, an instance of `javax.xml.xquery.XQSequence` must be accessed by only one thread. However, a restricted form of thread safety is supported for managing instances of `javax.xml.xquery.XQConnection`.

- An instance of `XQConnection` serves as a factory for creating instances of `XQExpression`, `XQPreparedExpression`, `XQItem`, `XQSequence`, `XQItemType`, and `XQSequenceType`. One thread can manage the creation of these objects for use by other threads. For example, `XQPreparedExpression` instances created in one thread by the same connection can be used in other threads. Each `XQPreparedExpression` instance, however, must be executed by only one thread. Any user-defined implementations of `oracle.xml.xquery.OXQEntityResolver` that are specified must be thread-safe when expressions from the same connection are evaluated concurrently.
- Method `XQConnection.close()` closes all `XQExpression` and `XQPreparedExpression` instances that were obtained from the connection. Closing those instances closes all `XQResultSequence` and `XQResultItem` instances obtained from the expressions. Method `XQConnection.close()` can be called while expressions obtained from the connection are being processed in other threads. In that case, all registered resources held by the expressions (such as `java.io.InputStream` and `java.io.Reader`) are closed. This contract assumes that all registered resources support a thread-safe close method. For example, many JDK implementations of `java.io.Closeable` satisfy this requirement. But, many implementations of `javax.xml.stream.XMLStreamReader` do not provide a thread-safe close method. Implementations without this support can give unpredictable results if they are closed while a second thread is still reading (see interface `oracle.xml.xquery.OXQCloseable` in *Oracle AI Database XML Java API Reference*).

> **See Also**
>
> *Oracle AI Database XML Java API Reference*, method `oracle.xml.xquery.OXQConnection.copyExpression(XQPreparedExpression)`

15.5 Performing Updates

XDK extends XQJ with the ability to execute updating queries. XML documents can be read as an instance of `javax.xml.xquery.XQItem`, and then modified using XQuery Update Facility extensions.

This feature is disabled by default. You can enable it by setting the update mode on the dynamic context to `oracle.xml.xquery.OXQConstants.UPDATE_MODE_ENABLED`.

Documents to be updated must be bound in deferred mode (see method `javax.xml.xquery.XQStaticContext.setBindingMode(int)` in *Oracle AI Database XML Java API Reference*). If the binding mode is not set to deferred, the input bindings are copied before query execution. Thus, only the copy is updated.

The example in this section shows how you can modify an XML document using the XQuery Update Facility.

Example 15-20 displays the contents of `configuration.xml`.

Example 15-21 displays the contents of `update.xq`.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Example 15-22 displays the contents of `configuration.xml` after an update.

Example 15-23 shows how execute the query `update.xq`.

In the example, these actions occur:

1. The XML file `configuration.xml` is read as an instance of `javax.xml.xquery.XQItem`.
2. The item is bound to the prepared expression for the query `update.xq`.
3. The query `update.xq` is executed.
4. The modified document is written to the file `configuration.xml`.

> **See Also**
>
> - *XQuery Update Facility 1.0*
> - *Oracle AI Database XML Java API Reference*, interface `oracle.xml.xquery.OXQDynamicContext`

Example 15-20 configuration.xml

```
<configuration>
  <property>
    <name>hostname</name>
    <value>example.com</value>
  </property>
  <property>
    <name>timeout</name>
    <value>1000</value>
  </property>
</configuration>
```

Example 15-21 update.xq

```
declare variable $doc external;

let $timeout := $doc/configuration/property[name eq "timeout"]
return
  replace value of node $timeout/value
  with 2 * xs:integer($timeout/value)
```

Example 15-22 Updated File configuration.xml

```
<configuration>
  <property>
    <name>hostname</name>
    <value>example.com</value>
  </property>
  <property>
    <name>timeout</name>
    <value>2000</value>
  </property>
</configuration>
```

Example 15-23 Executing the Updating Query update.xq

```
import java.io.FileInputStream;
import java.io.IOException;
import java.io.FileOutputStream;
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
import javax.xml.namespace.QName;
import javax.xml.xquery.XQConnection;
import javax.xml.xquery.XQConstants;
import javax.xml.xquery.XQException;
import javax.xml.xquery.XQItem;
import javax.xml.xquery.XQPreparedExpression;
import javax.xml.xquery.XQStaticContext;

import oracle.xml.xquery.OXQConstants;
import oracle.xml.xquery.OXQDataSource;
import oracle.xml.xquery.OXQView;

public class UpdateDocument {
    public static void main(String[] args) throws XQException, IOException {
        OXQDataSource ds = new OXQDataSource();
        XQConnection con = ds.getConnection();

        XQStaticContext ctx = con.getStaticContext();
        // Set the binding mode to deferred so the document
        // item is not copied when it is bound.
        ctx.setBindingMode(XQConstants.BINDING_MODE_DEFERRED);
        con.setStaticContext(ctx);

        FileInputStream input = new FileInputStream("configuration.xml");
        XQItem doc = con.createItemFromDocument(input, null, null);
        input.close();

        System.out.println("Before update: \n" + doc.getItemAsString(null));

        FileInputStream query = new FileInputStream("update.xq");
        XQPreparedExpression expr = con.prepareExpression(query);
        query.close();
        expr.bindItem(new QName("doc"), doc);
        // Enable updates (disabled by default)
        OXQView.getDynamicContext(expr).setUpdateMode(OXQConstants.UPDATE_MODE_ENABLED);
        expr.executeQuery();

        System.out.println("After update: \n" + doc.getItemAsString(null));

        // Write the modified document back to the file
        FileOutputStream out = new FileOutputStream("configuration.xml");
        doc.writeItem(out, null);

        expr.close();
        con.close();
    }
}
```

15.6 Oracle XQuery Functions and Operators

Oracle supports the standard XQuery functions and operators, as well as some Oracle-specific functions.

Oracle-specific XQuery functions use namespace `http://xmlns.oracle.com/xdk/xquery/function`. Namespace prefix `ora-fn` is predeclared, and the module is automatically imported.

Related Topics

- Standards and Specifications for the XQuery Processor for Java
  The standards and specifications to which the XDK XQuery processor for Java conforms are listed.

15.6.1 Oracle XQuery Functions for Duration, Date, and Time

You can manipulate durations, dates, and times in XQuery using Oracle XQuery functions.

Copyright © 1999, 2025, Oracle and/or its affiliates.

The Oracle XQuery functions are in namespace `http://xmlns.oracle.com/xdk/xquery/function`. Namespace prefix`ora-fn` is predeclared, and the module is automatically imported.

15.6.1.1 ora-fn:date-from-string-with-format

This Oracle XQuery function returns a new date value from a string according to a given pattern.

Signature

```
ora-fn:date-from-string-with-format($format as xs:string?,
                                    $dateString as xs:string?,
                                    $locale as xs:string*)
                                   as xs:date?

ora-fn:date-from-string-with-format($format as xs:string?,
                                    $dateString as xs:string?)
                                   as xs:date?
```

Parameters

`$format`: The pattern; see Format Argument

`$dateString`: An input string that represents a date

`$locale`: A one- to three-field value that represents the locale; see Locale Argument

Example

This example returns the specified date in the current time zone:

```
ora-fn:date-from-string-with-format("yyyy-MM-dd G", "2013-06-22 AD")
```

15.6.1.2 ora-fn:date-to-string-with-format

This Oracle XQuery function returns a date string with a given pattern.

Signature

```
ora-fn:date-to-string-with-format($format as xs:string?,
                                  $date as xs:date?,
                                  *$locale as xs:string?)
                                 as xs:string?

ora-fn:date-to-string-with-format($format as xs:string?,
                                  $date as xs:date?)
                                 as xs:string?
```

Parameters

`$format`: The pattern; see Format Argument

`$date`: The date

`$locale`: A one- to three-field value that represents the locale; see Locale Argument

Copyright © 1999, 2025, Oracle and/or its affiliates.

Example

This example returns the string `2013-07-15`:

```
ora-fn:date-to-string-with-format("yyyy-mm-dd", xs:date("2013-07-15"))
```

15.6.1.3 ora-fn:dateTime-from-string-with-format

This Oracle XQuery function returns a new date-time value from an input string, according to a given pattern.

Signature

```
ora-fn:dateTime-from-string-with-format($format as xs:string?,
                                        $dateTimeString as xs:string?,
                                        $locale as xs:string?)
                                       as xs:dateTime?

ora-fn:dateTime-from-string-with-format($format as xs:string?,
                                        $dateTimeString as xs:string?)
                                       as xs:dateTime?
```

Parameters

`$format`: The pattern; see Format Argument

`$dateTimeString`: The date and time

`$locale`: A one- to three-field value that represents the locale; see Locale Argument

Examples

This example returns the specified date and 11:04:00AM in the current time zone:

```
ora-fn:dateTime-from-string-with-format("yyyy-MM-dd 'at' hh:mm",
                                        "2013-06-22 at 11:04")
```

The next example returns the specified date and 12:00:00AM in the current time zone:

```
ora-fn:dateTime-from-string-with-format("yyyy-MM-dd G",
                                        "2013-06-22 AD")
```

15.6.1.4 ora-fn:dateTime-to-string-with-format

This Oracle XQuery function returns a date and time string with a given pattern.

Signature

```
ora-fn:dateTime-to-string-with-format($format as xs:string?,
                                      $dateTime as xs:dateTime?,
                                      $locale as xs:string?)
                                     as xs:string?

ora-fn:dateTime-to-string-with-format($format as xs:string?,
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
                                        $dateTime as xs:dateTime?)
                                      as xs:string?
```

Parameters

`$format`: The pattern; see Format Argument

`$dateTime`: The date and time

`$locale`: A one- to three-field value that represents the locale; see Locale Argument

Examples

This example returns the string `07 JAN 2013 10:09 PM AD`:

```
ora-fn:dateTime-to-string-with-format("dd MMM yyyy hh:mm a G",
                                      xs:dateTime("2013-01-07T22:09:44"))
```

The next example returns the string "01-07-2013":

```
ora-fn:dateTime-to-string-with-format("MM-dd-yyyy",
                                      xs:dateTime("2013-01-07T22:09:44"))
```

15.6.1.5 ora-fn:time-from-string-with-format

This Oracle XQuery function returns a new time value from an input string, according to a given pattern.

Signature

```
ora-fn:time-from-string-with-format($format as xs:string?,
                                    $timeString as xs:string?,
                                    $locale as xs:string?)
                                   as xs:time?

ora-fn:time-from-string-with-format($format as xs:string?,
                                    $timeString as xs:string?)
                                   as xs:time?
```

Parameters

`$format`: The pattern; see Format Argument

`$timeString`: The time

`$locale`: A one- to three-field value that represents the locale; see Locale Argument

Example

This example returns 9:45:22 PM in the current time zone:

```
ora-fn:time-from-string-with-format("HH.mm.ss", "21.45.22")
```

The next example returns 8:07:22 PM in the current time zone:

```
fn-bea:time-from-string-with-format("hh:mm:ss a", "8:07:22 PM")
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

15.6.1.6 ora-fn:time-to-string-with-format

This Oracle XQuery function returns a time string with a given pattern.

Signature

```
ora-fn:time-to-string-with-format($format as xs:string?,
                                  $time as xs:time?,
                                  $locale as xs:string?)
                                 as xs:string?

ora-fn:time-to-string-with-format($format as xs:string?, $time as xs:time?)
as xs:string?
```

Parameters

`$format`: The pattern; see Format Argument

`$time`: The time

`$locale`: A one- to three-field value that represents the locale; see Locale Argument

Examples

This example returns the string "10:09 PM":

```
ora-fn:time-to-string-with-format("hh:mm a", xs:time("22:09:44"))
```

The next example returns the string "22:09 PM":

```
ora-fn:time-to-string-with-format("HH:mm a", xs:time("22:09:44"))
```

15.6.1.7 Format Argument

The `$format` argument identifies the various fields that compose a date or time value.

15.6.1.8 Locale Argument

The `$locale` represents a specific geographic, political, or cultural region.

It is defined by up to three fields:

1. **Language code**: The ISO 639 alpha-2 or alpha-3 language code, or the registered language subtags of up to eight letters. For example, `en` for English and `ja` for Japanese.
2. **Country code**: The ISO 3166 alpha-2 country code or the UN M.49 numeric-3 area code. For example, `US` for the United States and `029` for the Caribbean.
3. **Variant**: Indicates a variation of the locale, such as a particular dialect. Order multiple values in order of importance and separate them with an underscore (_). These values are case sensitive.

Copyright © 1999, 2025, Oracle and/or its affiliates.

> **See Also**
>
> - Class Locale in *Java Standard Edition 7 Reference*

15.6.2 Oracle XQuery Functions for Strings

You can manipulate strings in XQuery using Oracle XQuery functions.

The Oracle XQuery functions are in namespace `http://xmlns.oracle.com/xdk/xquery/function`. Namespace prefix`ora-fn` is predeclared, and the module is automatically imported.

15.6.2.1 ora-fn:pad-left

Adds padding characters to the left of a string to create a fixed-length string. If the input string exceeds the specified size, then it is truncated to return a substring of the specified length. The default padding character is a space (ASCII 32).

Signature

```
ora-fn:pad-left($str as xs:string?,
                $size as xs:integer?,
                $pad as xs:string?)
               as xs:string?

ora-fn:pad-left($str as xs:string?,
                $size as xs:integer?)
               as xs:string?
```

Parameters

`$str`: The input string

`$size`: The desired fixed length, which is obtained by adding padding characters to `$str`

`$pad`: The padding character

If either argument is an empty sequence, then the function returns an empty sequence.

Examples

This example prefixes "`01`" to the input string up to the maximum of six characters. The returned string is "`010abc`". The function returns one complete and one partial pad character.

```
ora-fn:pad-left("abc", 6, "01")
```

The example returns only "`ab`" because the input string exceeds the specified fixed length:

```
ora-fn:pad-left("abcd", 2, "01")
```

This example prefixes spaces to the string up to the specified maximum of six characters. The returned string has a prefix of two spaces: "`  abcd`":

```
ora-fn:pad-left("abcd", 6)
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

The next example returns only "ab" because the input string exceeds the specified fixed length:

```
ora-fn:pad-left("abcd", 2)
```

15.6.2.2 ora-fn:pad-right

Adds padding characters to the right of a string to create a fixed-length string. If the input string exceeds the specified size, then it is truncated to return a substring of the specified length. The default padding character is a space (ASCII 32).

Signature

```
ora-fn:pad-right($str as xs:string?,
                 $size as xs:integer?,
                 $pad as xs:string?)
                as xs:string?

ora-fn:pad-right($str as xs:string?,
                 $size as xs:integer?)
                as xs:string?
```

Parameters

$str: The input string

$size: The desired fixed length, which is obtained by adding padding characters to $str

$pad: The padding character

If either argument is an empty sequence, then the function returns an empty sequence.

Examples

This example appends "01" to the input string up to the maximum of six characters. The returned string is "abc010". The function returns one complete and one partial pad character.

```
ora-fn:pad-right("abc", 6, "01")
```

This example returns only "ab" because the input string exceeds the specified fixed length:

```
ora-fn:pad-right("abcd", 2, "01")
```

This example appends spaces to the string up to the specified maximum of six characters. The returned string has a suffix of two spaces: "abcd ":

```
ora-fn:pad-right("abcd", 6)
```

The next example returns only "ab" because the input string exceeds the specified fixed length:

```
ora-fn:pad-right("abcd", 2)
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

15.6.2.3 ora-fn:trim

Removes any leading or trailing white space from a string.

Signature

```
ora-fn:trim($input as xs:string?) as xs:string?
```

Parameters

`$input`: The string to trim. If `$input` is an empty sequence, then the function returns an empty sequence. Other data types trigger an error.

Example

This example returns the string "`abc`":

```
ora-fn:trim("  abc  ")
```

15.6.2.4 ora-fn:trim-left

Removes any leading white space.

Signature

```
ora-fn:trim-left($input as xs:string?) as xs:string?
```

Parameters

`$input`: The string to trim. If `$input` is an empty sequence, then the function returns an empty sequence. Other data types trigger an error.

Example

This example removes the leading spaces and returns the string "`abc    `":

```
ora-fn:trim-left("    abc    ")
```

15.6.2.5 ora-fn:trim-right

Removes any trailing white space.

Signature

```
ora-fn:trim-right($input as xs:string?) as xs:string?
```

Parameters

`$input`: The string to trim. If `$input` is an empty sequence, then the function returns an empty sequence. Other data types trigger an error.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Example

This example removes the trailing spaces and returns the string " abc":

```
ora-fn:trim-left("    abc    ")
```

15.7 Standards and Specifications for the XQuery Processor for Java

The standards and specifications to which the XDK XQuery processor for Java conforms are listed.

- *XQuery 3.0: An XML Query Language*

> **Note**
>
> All XQuery 1.0 level features are supported. XQuery 3.0 level features are supported except for the following: FLWOR window clause, FLWOR count clause, namespace constructors, decimal format declarations, `fn:format-number`, `fn:format-integer`, `fn:format-date`, `fn:format-time`, `fn:path`, and higher order XQuery functions.

- *XQuery Update Facility 1.0*
- *XQueryX 3.0*, the XML syntax for XQuery 3.0
- *JSR-000225 XQuery API for Java*

> **Note**
>
> The XDK XQuery processor for Java is not interoperable with other XQJ implementations, including the Oracle XQJ implementation for Oracle XML DB. (See JSR-225: XQuery API for Java for the meaning of *interoperable*.)

15.7.1 Optional XQuery Features

The XQuery specification defines certain features as optional. Links are provided to the standards that specify those that are supported by XDK.

These are the optional features supported by XDK:

- Schema Import
- Validate Expressions
- Static Typing
- Static Typing for Update
- Modules
- Serialization

Copyright © 1999, 2025, Oracle and/or its affiliates.

15.7.2 Implementation-Defined Items

The XQJ and XQuery specifications leave the definition of certain aspects up to the implementation. The implementation-defined items for XDK are described briefly.

Table 15-2 summarizes the XQJ implementation-defined items.

Table 15-2 XQJ Implementation-Defined Items

Description	Behavior
Class name of `XQDataSource` implementation	`oracle.xml.xquery.OXQDataSource`.
Properties defined on `OXQDataSource`	None. The username and password are silently ignored.
JDBC connection support	JDBC connections are not supported.
Commands	Not supported.
Cancelling of query execution with method `XQPreparedExpression.cancel()`	Yes.
Serialization	Yes.
Additional StAX or SAX events	None.
User-defined schema types	Yes.
Node identity, document order, and full-node context preservation when a node is bound to an external variable	Not preserved.
Login timeout	Not supported.
Transactions	Not supported. An exception is thrown if a transaction method is called.
`XQItemAccessor.getNodeUri()` method behavior if the input node is not a document node	Exception.
`XQItemType.getTypeName()` method for anonymous types	A unique name.
`XQItemType.getSchemaURI()` method	The schema URI is returned when a type is created from XQJ. No otherwise.
`XQDataFactory.createItemFromDocument()` and `bindDocument()` methods if the input is not a well-formed XML document	Exception.
Additional error codes returned by class `XQQueryException`	The qualified names of Oracle-specific error codes are in the namespace `http://xmlns.oracle.com/xdk/xquery/errors`.
`ConnectionPoolXQDataSource`, `PooledXQConnection`, `XQConnectionEvent`, `XQConnectionEventListener` interfaces	No.
`XQDataSource.getConnection(java.sql.Connection)`	JDBC connections are not supported. An exception is thrown if this method is called.
`XQDataSource.getConnection(java.lang.String, java.lang.String)`	Same as `getConnection()`. Parameters are ignored.

Copyright © 1999, 2025, Oracle and/or its affiliates.

> **Note**
>
> XDK support for the features in Table 15-2 differs from the Oracle XML DB support for them.

Table 15-3 summarizes the XQuery implementation-defined items.

Table 15-3 XQuery Implementation-Defined Items

Item	Behavior
The version of Unicode that is used to construct expressions	Depends on the version of Java used. Different Java versions support different versions of Unicode.
The statically-known collations	Unicode codepoint collation and collations derived from classes `java.text.Collator` or `oracle.i18n.text.OraCollator`.
The implicit time zone.	Uses the default time zone, as determined by method `java.util.Calendar.getInstance()`.
The circumstances in which warnings are raised, and the ways in which warnings are handled	None.
The method by which errors are reported to the external processing environment	Exception `javax.xml.xquery.XQException`.
Whether the implementation is based on the rules of XML 1.0 and XML Names, or the rules of XML 1.1 and XML Names 1.1	1.0.
Any components of the static context or dynamic context that are overwritten or augmented by the implementation	See Table 15-5.
Which of the optional axes are supported by the implementation, if the Full-Axis Feature is not supported	Full support.
The default handling of empty sequences returned by an ordering key (sortspec) in an order by clause (empty least or empty greatest)	`least`.
The names and semantics of any extension expressions (pragmas) recognized by the implementation	None.
The names and semantics of any option declarations recognized by the implementation	None.
Protocols (if any) by which parameters can be passed to an external function, and the result of the function can be returned to the invoking query	Defined by XQJ.
The process by which the specific modules to be imported by a module import are identified, if the Module feature is supported (includes processing of location hints, if any)	Entity resolvers. See XQJ Entity Resolution.
Any static typing extensions supported by the implementation, if the Static Typing feature is supported	Strict mode (based on subtype) and optimistic mode (based on type intersection). Optimistic mode is the default.
The means by which serialization is invoked, if the Serialization feature is supported	Defined by XQJ.
The default values for the `byte-order-mark`, `encoding`, `media-type`, `normalization-form`, `omit-xml-declaration`, `standalone`, and `version` parameters, if the Serialization feature is supported	See the interface `oracle.xml.xquery.OXQSerializationParameters` in *Oracle AI Database XML Java API Reference*.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 15-3 (Cont.) XQuery Implementation-Defined Items

Item	Behavior
Limits on ranges of values for various data types.	Decimal and integer values have arbitrary precision.
The signatures of functions provided by the implementation or via an implementation-defined API (see the XQuery standard, section 2.1.1, *Static Context*).	See Oracle XQuery Functions and Operators.
Any environment variables provided by the implementation.	Entity resolvers. See XQJ Entity Resolution.
Any rules used for static typing (see the XQuery standard, section 2.2.3.1, *Static Analysis Phase*).	Defaults to optimistic, pessimistic configurable.
Any serialization parameters provided by the implementation (see the XQuery standard, section 2.2.4 *Serialization*).	See `OXQSerializationParameters`. See *Oracle AI Database XML Java API Reference*.
The means by which the location hint for a serialization parameter document identifies the corresponding XDM instance (see the XQuery standard, section 2.2.4, *Serialization*).	Entity resolvers, `OXQEntityKind.DOCUMENT`. See XQJ Entity Resolution.
What error, if any, is returned if an external function's implementation does not return the declared result type (see the XQuery standard, section 2.2.5, *Consistency Constraints*).	`XPTY0004`.
Any annotations defined by the implementation, and their associated behavior (see the XQuery standard, section 4.15, *Annotations*).	You can use `%ora-fn:context-item` to allow the context item of a function caller to be implicitly passed as the first argument to the function. For example, this query evaluates to `e`, `f`, `g`: `declare %ora-fn:context-item function local:foo($arg as node()) { node-name($arg) }; (<e/>, <f/>)/local:foo(), local:foo(<g/>)`
Any function assertions defined by the implementation.	None.
The effect of function assertions understood by the implementation on section 2.5.6.3. The judgment subtype-assertions (AnnotationsA, AnnotationsB) .	Not applicable.
Any implementation-defined variables defined by the implementation. (see the XQuery standard, section 3.1.2, *Variable References*).	None.
The ordering associated with `fn:unordered` in the implementation (see the XQuery standard, section 3.11, *Ordered and Unordered Expressions*).	It does not change the order of the input sequence.
Any additional information provided for `try/catch` by variable `err:additional` (see the XQuery standard, section 3.15, *Try/Catch Expressions*).	None.
The default boundary-space policy (see the XQuery standard, section 4.3, *Boundary-space Declaration*).	`strip`.
The default collation (see the XQuery standard, section 4.4, *Default Collation Declaration*).	Unicode.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 15-3 (Cont.) XQuery Implementation-Defined Items

Item	Behavior
The default base URI (see the XQuery standard, section 4.5, *Base URI Declaration*).	None.

Table 15-4 summarizes the XQuery Update Facility implementation-defined items.

Table 15-4 XQuery Update Facility Implementation-Defined Items

Item	Behavior
The revalidation modes that are supported by this implementation.	`skip`.
The default revalidation mode for this implementation.	`skip`.
The mechanism (if any) by which an external function can return an XDM instance, or a pending update list, or both to the invoking query.	Returning a pending update list from an external function is not supported.
The semantics of `fn:put()`, including the kinds of nodes accepted as operands by this function.	Any node type is accepted. Storage of the node is determined by the entity resolver. See class `oracle.xml.xquery.OXQEntity` in *Oracle AI Database XML Java API Reference*, specifically the documentation for entity kind `UPD_PUT`.

Table 15-5 summarizes the default initial values for the static context.

Table 15-5 Default Initial Values for the Static Context

Context Component	Default Value

Statically known namespaces	`err=http://w3.org/2005/xqt-errors`
	`fn=http://www.w3.org/2005/xpath-functions`
	`local=http://www.w3.org/2005/xquery-local-functions`
	`math=http://www.w3.org/2005/xpath-functions/math`
	`ora-ext=http://xmlns.oracle.com/xdk/xquery/extension`
	`ora-java=http://xmlns.oracle.com/xdk/xquery/java`
	`ora-xml=http://xmlns.oracle.com/xdk/xquery/xml`
	`ora-fn=http://xmlns.oracle.com/xdk/xquery/function`
	`output=http://www.w3.org/2010/xslt-xquery-serialization`
	`xml=http://www.w3.org/XML/1998/namespace`
	`xs=http://www.w3.org/2001/XMLSchema`
	`xsi=http://www.w3.org/2001/XMLSchema-instance`
	Prefixes that begin with `ora-` are reserved for use by Oracle. Additional prefixes that begin with `ora-` may be added to the default statically known namespaces in a future release.

Default element/type namespace	No namespace.
Default function namespace	`fn`.
In-scope schema types	Built-in types in `xs`.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 15-5 (Cont.) Default Initial Values for the Static Context

Context Component	Default Value
In-scope element declarations	None.
In-scope attribute declarations	None.
In-scope variables	None.
Context item static type	`item()`.
Function signatures	Functions in namespace `fn`, and constructors for built-in atomic types.
Statically known collations	Unicode codepoint collation and collations derived from class `java.text.Collator` or `oracle.i18n.text.OraCollator`.
Default collation	Unicode codepoint collation: `http://www.w3.org/2005/xpath-functions/collation/codepoint`.
Construction mode	`preserve`.
Ordering mode	`ordered`.
Default order for empty sequences	`least`.
Boundary-space policy	`strip`.
Copy-namespaces mode	`preserve`, `no-inherit`.
Base URI	As defined in the standard.
Statically known documents	None.
Statically known collections	None.
Statically known default collection type	`node()*`.
Serialization parameters	Same as the defaults for `OXQSerializationParameters`. See *Oracle AI Database XML Java API Reference*.

Copyright © 1999, 2025, Oracle and/or its affiliates.

16

Using XQuery API for Java to Access Oracle XML DB

An explanation is given of how to use the XQuery API for Java (XQJ) to access Oracle XML DB.

16.1 Introduction to Oracle XML DB Support for XQJ

XQuery API for Java (XQJ), also known as JSR-225, provides an industry-standard way for Java programs to access Extensible Markup Language (XML) data using XQuery. It lets you evaluate XQuery expressions against XML data sources and process the results as XML data.

Oracle provides two XQuery engines for evaluating XQuery expressions: one in Oracle XML DB, for use with XML data in the database, and one in Oracle XML Developer's Kit (XDK), for use with XML data outside the database. (See Using the XQuery Processor for Java for information about the XQuery engine for XDK).

Oracle provides two different XQJ implementations for accessing these two XQuery engines. Both implementations are part of XDK, enabling you to use XDK to access XML data with a standard XQJ API whether that data resides in the database or elsewhere.

The queries executed by XQJ are written in standard World Wide Web Consortium (W3C) XQuery 1.0 language, as supported by Oracle XML DB. A typical use case for this feature is to access XML data stored in remote databases (in Oracle XML DB) from a local Java program.

General information about XQuery and XQJ is documented outside of this document.

See Also

- *Oracle XML DB Developer's Guide* for more information about Oracle XML DB, including details about XQuery capabilities and support in Oracle XML DB
- `XQuery Packages` in *Oracle AI Database XML Java API Reference* for the related API documentation
- *JSR-000225 XQuery API for Java*, which is very concrete and has understandable examples

16.1.1 Prerequisites for Using XQJ to Access Oracle XML DB

You need Java Runtime Environment 1.6 to use XQJ with Oracle XML DB. You also need certain Java Archive (JAR) files to be either in your `CLASSPATH` environment variable or passed using command-line option `classpath`.

The JAR files are as follows:

- The required JAR files listed in Introduction to the XQuery Processor for Java
- `jdbc/lib/ojdbc6.jar`

Copyright © 1999, 2025, Oracle and/or its affiliates.

- `rdbms/jlib/xdb6.jar`

The directory paths for these JAR files are relative to the `ORACLE_HOME` directory of your database installation.

16.2 Examples: Using XQJ to Query Oracle XML DB

Examples here show how you can use XQJ to query and retrieve data in Oracle XML DB.

Example 16-1 shows how to use XQJ to query data from a table in Oracle XML DB. It uses the `WAREHOUSES` table in the Order Entry (OE) database sample schema. The OE sample schema contains XML documents with warehouse information in the `WAREHOUSES` table. The `WAREHOUSES` table contains an XMLType column `warehouse_spec` and other columns. (See the discussion about standard database schemas in *Oracle XML DB Developer's Guide* for more information about the data used in this example.)

Specifically, Example 16-1 shows how to perform these steps:

1. Get an XQJ connection to Oracle XML DB.

   Every program using XQJ to connect to Oracle XML DB must first create an `OXQDDataSource` object. Then, `OXQDDataSource` must be initialized with the required property values before getting an XQJ connection to the Oracle XML DB instance.
2. Prepare an XQuery expression.
3. Submit the XQuery expression for evaluation.
4. Print each item in the resulting XQuery sequence.

In Example 16-1, the XQuery expression accesses the `WAREHOUSES` table through the use of the Universal Resource Identifier (URI) scheme `oradb`. (See the discussion about the URI scheme `oradb` in *Oracle XML DB Developer's Guide* for more information about using XQuery with XML DB to query table or view data.)

Example 16-1 also shows how to bind external variable values in XQJ. The query has an external variable `$x`, which is used to filter the returned rows from the `WAREHOUSES` table, by `WAREHOUSE_ID`.

Example 16-1 generates this output (reformatted for better readability):

```
<Warehouse><Building>Owned</Building><Area>25000</Area><Docks>2</Docks>
 <DockType>Rearload</DockType><WaterAccess>Y</WaterAccess>
 <RailAccess>N</RailAccess><Parking>Street</Parking>
 <VClearance>10 ft</VClearance></Warehouse>

<Warehouse><Building>Rented</Building><Area>50000</Area><Docks>1</Docks>
 <DockType>Sideload</DockType><WaterAccess>Y</WaterAccess>
 <RailAccess>N</RailAccess><Parking>Lot</Parking>
 <VClearance>12 ft</VClearance></Warehouse>
```

Example 16-2 shows how to use XQJ to retrieve data from the Oracle XML DB repository. This example assumes that two files, `depts.xml` and `emps.xml`, have been uploaded into the XML DB repository under the folder `/public`. For example, you can use FTP to upload the two files into the Oracle XML DB repository. (See the discussion about using the Oracle XML DB repository in *Oracle XML DB Developer's Guide* for more information about storing data in and using the Oracle XML DB Repository.)

The content of `depts.xml` is:

`depts.xml:`

Copyright © 1999, 2025, Oracle and/or its affiliates.

```
<?xml version="1.0"?>
     <depts>
       <dept deptno="10" dname="Administration"/>
       <dept deptno="20" dname="Marketing"/>
       <dept deptno="30" dname="Purchasing"/>
     </depts>
```

The content of `emps.xml` is:

```
emps.xml:

<?xml version="1.0"?>
   <emps>
    <emp empno="1" deptno="10" ename="John"  salary="21000"/>
       <emp empno="2" deptno="10" ename="Jack" salary="310000"/>
       <emp empno="3" deptno="20" ename="Jill" salary="100001"/>
     </emps>
```

You can use the `fn:doc` and `fn:collection` functions to query the data in the Oracle XML DB repository with XQuery. Example 16-2 shows how to use the `fn:doc` function within XQuery to access the repository. (See the discussion about querying XML data in the Oracle XML DB repository in *Oracle XML DB Developer's Guide* for more information about using these XQuery functions.)

Example 16-2 generates this output:

```
<emp ename="Jack" dept="Administration"/>
<emp ename="Jill" dept="Marketing"/>
```

Example 16-1 Using XQJ to Query an XML DB Table with XQuery

```
import oracle.xml.xquery.xqjdb.OXQDDataSource;

import javax.xml.xquery.XQItemType;
import javax.xml.xquery.XQResultSequence;
import javax.xml.xquery.XQConnection;
import javax.xml.xquery.XQPreparedExpression;
import javax.xml.namespace.QName;

public class example1
{
  public static void main(String argv[])
  {
    try
    {
      // Create a new OXQDDataSource for connecting to Oracle XML DB
      OXQDDataSource oxqDS = new OXQDDataSource();
      // Set appropriate connection information for the database instance.
      // Must use the thin driver
      oxqDS.setProperty("driver", "jdbc:oracle:thin");
      oxqDS.setProperty("dbusername", "oe");
      oxqDS.setProperty("dbpassword", "oe");
      // Machine hostname
      oxqDS.setProperty("dbserver", "myserver");
      // Database instance port number
      oxqDS.setProperty("dbport", "6479");
      // Database instance port number
      oxqDS.setProperty("serviceName", "mydbinstance");
      XQConnection conn = oxqDS.getConnection();
      XQItemType  itemTypeInt = conn.createAtomicType(XQItemType.XQBASETYPE_INT);
      XQPreparedExpression expr = conn.prepareExpression("declare variable $x as
        xs:int external; for $i in fn:collection('oradb:/OE/WAREHOUSES') where
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
          $i/ROW/WAREHOUSE_ID < $x return $i/ROW/WAREHOUSE_SPEC/Warehouse");
      expr.bindInt(new QName("x"), 3, itemTypeInt);
      XQResultSequence xqSeq = expr.executeQuery();
      while (xqSeq.next())
         System.out.println (xqSeq.getItemAsString(null));
     }
     catch (Exception e)
     {
      e.printStackTrace();
     }
  }
}
```

Example 16-2 Using XQJ to Query the XML DB Repository with XQuery

```
import oracle.xml.xquery.xqjdb.OXQDDataSource;

import javax.xml.xquery.XQItemType;
import javax.xml.xquery.XQResultSequence;
import javax.xml.xquery.XQConnection;
import javax.xml.xquery.XQPreparedExpression;
import javax.xml.namespace.QName;

public class example2
{
  public static void main(String argv[])
  {
    try
    {
      // Create a new OXQDDataSource for connecting to Oracle XML DB
      OXQDDataSource oxqDS = new OXQDDataSource();
      // Set appropriate connection information for the database instance.
      // Must use the thin driver
      oxqDS.setProperty("driver", "jdbc:oracle:thin");
      oxqDS.setProperty("dbusername", "oe");
      oxqDS.setProperty("dbpassword", "oe");
      // Machine hostname
      oxqDS.setProperty("dbserver", "myserver");
      // Database instance port number
      oxqDS.setProperty("dbport", "6479");
      // Database instance port number
      oxqDS.setProperty("serviceName", "mydbinstance");
      XQConnection conn = oxqDS.getConnection();
      XQPreparedExpression expr = conn.prepareExpression("for $e in
        doc(\"/public/emps.xml\")/emps/emp let $d :=
        doc(\"/public/depts.xml\")//dept[@deptno = $e/@deptno]/@dname where
        $e/@salary > 100000 order by $e/@empno return
        <emp ename=\"{$e/@ename}\" dept=\"{$d}\"/>");
      XQResultSequence xqSeq = expr.executeQuery();
      while (xqSeq.next())
         System.out.println (xqSeq.getItemAsString(null));
     }
     catch (Exception e)
     {
      e.printStackTrace();
     }
  }
}
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

16.3 XQJ Support for Oracle XML DB

The two Oracle XQJ implementations differ in some respects. Oracle XML DB support for XQJ is described.

Using the XQuery Processor for Java provides information about using XQJ to access the mid-tier XQuery engine.

Table 16-1 describes the OXQDDataSource properties to be used for connection to Oracle XML DB. To create an XQJ connection to Oracle XML DB, you must set the values for these properties. You must set either the `dbname` or the `serviceName` property value, and all the other OXQDDataSource property values listed in Table 16-1.

Table 16-1 OXQDDataSource Properties

Property	Value	Get Method	Set Method
`driver`	`jdbc:oracle:thin`	`getDriver`	`setDriver`
`dbusername`	Database schema (user) name	`getDBUserName`	`setDBUserName`
`dbpassword`	Password for database schema	`getDBPassword`	`setDBPassword`
`dbserver`	Host name for the database instance	`getDBServer`	`setDBServer`
`dbport`	Port number of the database instance for XQJ connection	`getDBPort`	`setDBPort`
`dbname`	Database instance name (service id)[1]	`getDBName`	`setDBName`
`serviceName`	Service name[1]	`getServiceName`	`setServiceName`

[1] You can identify the database using either the service id or the service name.

Table 16-2 describes the Oracle XML DB support for optional XQJ features.

> **Note**
>
> Oracle XML DB support for some XQJ features differs from their support by the mid-tier XQuery engine. In particular, the Oracle XML DB XQJ implementation does not support the use of user-defined types.

Table 16-2 Oracle XML DB Support for Optional XQJ Features

XQJ Feature	Oracle XML DB Support
Class name of `XQDataSource` implementation	`oracle.xml.xquery.xqjdb.OXQDDataSource`
JDBC connections	Not supported.
Properties defined on `OXQDDataSource` (connection information)	See Table 16-1.
Commands	Not supported.
`XQPreparedExpression.cancel` (cancelling of query execution)	Not supported.
Serialization	Only parameter `method` with value `xml` and parameter `encoding` with value `UTF-8` or `UTF-16`.
Additional StAX and SAX events	Not supported.
User-defined schema types	Not supported.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 16-2 (Cont.) Oracle XML DB Support for Optional XQJ Features

XQJ Feature	Oracle XML DB Support
Node identity, document order, and full-node context preservation when a node is bound to an external variable	Not supported.
Login timeout	Not supported.
Transactions	Not supported.
Behavior of `XQItemAccessor` method `getNodeUri()` when the input node is not a document node	Return `NULL`.
Behavior of `XQItemType` method `getTypeName()` for anonymous types	Return `false`.
Behavior of `XQItemType` method `getSchemaURI()`	Return `NULL` or the schema URI provided during type creation. Currently, the Oracle XML DB XQJ implementation does not use the schema URI to get type information, and user-defined types are not supported.
Behavior of `XQDataFactory` methods `createItemFromDocument()` and `bindDocument()` if the input is not a well-formed XML document	Raise an exception.
Additional error codes returned from `XQQueryException`	Not supported.
Interfaces `ConnectionPoolXQDataSource`, `PooledXQConnection`, `XQConnectionEvent`, `XQConnectionEventListener`	Not supported.
`XQDataSource.getConnection( java.sql.Connection)`	Not supported. (JDBC connections are not supported.)
`XQDataSource.getConnection( java.lang.String, java.lang.String)`	Same as `getConnection()` with no arguments: the arguments are ignored.

See Also

Oracle XML DB Developer's Guide for information about using the XQuery language with Oracle XML DB

16.3.1 Other Oracle XML DB XQJ Support Limitations

The limitations of Oracle XML DB support for XQJ are described. None of them apply to mid-tier XQuery engine support for XQJ.

Oracle XML DB support for XQJ is limited in these ways:

- All Oracle XML DB XQuery support limitations apply to Oracle XML DB support for XQJ as well.
- Only the XDK Document Object Model (DOM) is supported. Use of any other DOM can cause errors.

Copyright © 1999, 2025, Oracle and/or its affiliates.

- Do not expect the Oracle XML DB XQJ implementation to be interoperable with another XQJ implementation, including the XDK Java implementation of XQJ. (See the XQJ standard (JSR-225) for the meaning of "interoperable".)
- `XQDataSource` methods `getLogWriter` and `setLogWriter` have no effect (they are ignored).
- `XQStaticContent` methods `getBoundarySpacePolicy`, `setBoundarySpacePolicy`, `getDefaultCollation`, and `setDefaultCollation` have no effect (they are ignored).
- The copy namespaces mode for `XQStaticContent` methods `setCopyNamespacesModPreserve` and `setCopyNamespacesModeInherit` has no effect (it is ignored). The values used are always `preserve` and `inherit`, respectively.
- Use of `XQDynamicContext` methods to bind `DocumentFragment` objects is not supported.
- Values of type `xs:duration` are not supported. Using an `XQDynamicContext` method to bind `xs:duration`, or accessing an `xs:duration` value, raises an error.
- The year of a `xs:date`, `xs:dateTime`, `xs:gYear`, and `xs:gYearMonth` value must be from -4712 to 9999, inclusive. Using a year outside this range can raise an error or produce unpredictable results.

16.4 XQJ Performance Considerations for Use with Oracle XML DB

To fetch a sequence of items from the database, use `XQResultSequence` method `next()` to retrieve a single item at a time; then use an `XQItemAccessor` method to fetch all data corresponding to that item.

This provides better performance than using these whole-sequence fetch methods, which each materialize the entire sequence before returning any data.

- `getSequenceAsStream()`
- `getSequenceAsString(java.util.Properties props)`
- `writeSequence(java.io.OutputStream os, java.util.Properties props)`
- `writeSequence(java.io.Writer ow, java.util.Properties props)`
- `writeSequenceToResult(javax.xml.transform.Result result)`
- `writeSequenceToSAX(org.xml.sax.ContentHandler saxhdlr)`

For example, if you invoke `getSequenceAsStream()`, all of the XQuery result sequence data is fetched from the database before the `XMLStreamReader` instance that is built from it is returned to your program.

Be aware also that items themselves are not streamable: the item accessor methods always materialize an entire item before outputting any part of it.

For inputting, all bind methods defined on `XQDynamicContext` fully materialize the input data before passing it to the database.

For example, when you invoke `bindDocument(javax.xml.namespace.QName varName, javax.xml.stream.XMLStreamReader value, XQItemType type)`, all the data that is referenced by the input `XMLStreamReader` instance is processed before the external XQuery variable is bound to it.

Copyright © 1999, 2025, Oracle and/or its affiliates.

17

Using the XML Schema Processor for Java

Topics here cover how to use the Extensible Markup Language (XML) schema processor for Java.

17.1 Introduction to XML Validation

Topics cover the different techniques for XML validation.

17.1.1 Prerequisites for Using the XML Schema Processor for Java

Prerequisites for using the XML schema processor are covered.

This section assumes that you have working knowledge of these technologies:

- document type definition (DTD). An XML document type definition (DTD) defines the legal structure of an XML document.
- XML Schema language. XML Schema defines the legal structure of an XML document.

To learn more about these technologies, consult the XML resources in Related Documents.

17.1.2 Standards and Specifications for the XML Schema Processor for Java

XML Schema is a World Wide Web Consortium (W3C) standard.

The Oracle XML Schema processor supports the W3C XML Schema specifications:

- *XML Schema Part 0: Primer*
- *XML Schema Part 1: Structures*
- *XML Schema Part 2: Datatypes*

Related Topics

- Oracle XML Developer's Kit Standards
  A description is given of the Oracle XML Developer's Kit (XDK) standards.

17.1.3 XML Validation with DTDs

Document type definition (DTDs) were originally developed for SGML. XML DTDs are a subset of those available in SGML and provide a mechanism for declaring constraints on XML markup. XML DTDs enable the specification of:

- Which elements can be in your XML documents.
- The content model of an XML element, that is, whether the element contains only data or has a set of subelements that defines its structure. DTDs can define whether a subelement is optional or mandatory and whether it can occur only once or multiple times.

Copyright © 1999, 2025, Oracle and/or its affiliates.

- Attributes of XML elements. DTDs can also specify whether attributes are optional or mandatory.
- Entities that are legal in your XML documents.

An XML DTD is not itself written in XML, but is a context-independent grammar for defining the structure of an XML document. You can declare a DTD in an XML document itself or in a separate file from the XML document.

Validation is the process by which you verify an XML document against its associated DTD, ensuring that the structure, use of elements, and use of attributes are consistent with the definitions in the DTD. Thus, applications that handle XML documents can assume that the data matches the definition.

Using XDK, you can write an application that includes a validating XML parser; that is, a program that parses and validates XML documents against a DTD. Depending on its implementation, a validating parser may:

- Either stop processing when it encounters an error, or continue.
- Either report warnings and errors as they occur or in summary form at the end of processing.
- Enable or disable validation mode

  Most processors can enable or disable validation mode, but they must still process entity definitions and other constructs of DTDs.

17.1.3.1 DTD Samples in XDK

An example DTD is shown, together with an example XML document that conforms to that DTD.

Example 17-1 shows the contents of a DTD named `family.dtd`, which is located in `$ORACLE_HOME/xdk/demo/java/parser/common/`. The `<ELEMENT>` tags specify the legal nomenclature and structure of elements in the document, whereas the `<ATTLIST>` tags specify the legal attributes of elements.

Example 17-2 shows the contents of an XML document named `family.xml`, which is also located in `$ORACLE_HOME/xdk/demo/java/parser/common/`. The `<!DOCTYPE>` element in `family.xml` specifies that this XML document conforms to the external DTD named `family.dtd`.

Example 17-1 family.dtd

```
<?xml version="1.0" encoding="UTF-8"?>
<!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>
```

Example 17-2 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>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
<member memberid="m4" mom="m1" dad="m2">Jim</member>
</family>
```

17.1.4 XML Validation with XML Schemas

Concepts involving validation using XML schemas are introduced.

The XML Schema language, also known as XML Schema Definition, was created by the W3C to use XML syntax to describe the content and the structure of XML documents. An XML schema is an XML document written in the XML Schema language. An XML schema document contains rules describing the structure of an input XML document, called an instance document. An instance document is valid if and only if it conforms to the rules of the XML schema.

The XML Schema language defines such things as:

- Which elements and attributes are legal in the instance document
- Which elements can be children of other elements
- The order and number of child elements
- Data types for elements and attributes
- Default and fixed values for elements and attributes

A validating XML parser tries to determine whether an instance document conforms to the rules of its associated XML schema. Using XDK you can write a validating parser that performs this schema validation. Depending on its implementation, a validating parser may:

- Either stop processing when it encounters an error, or continue.
- Either report warnings and errors as they occur or in summary form at the end of processing.

The processor must consider entity definitions and other constructs that are defined in a DTD that is included by the instance document. The XML Schema language does not define what must occurs when an instance document includes both an XML schema and a DTD. Thus, the behavior of the application in such cases depends on the implementation.

17.1.4.1 XML Schema Samples in XDK

A sample XML document is shown which contains a purchase report that describes parts that have been ordered in different regions. This document is located at `$ORACLE_HOME/xdk/demo/java/schema/report.xml`. An XML schema document, `report.xsd`, which you can use to validate `report.xml`, is also shown.

Among other things, the XML schema defines the names of the elements that are legal in the instance document and the type of data that the elements can contain.

Example 17-3 report.xml

```
<purchaseReport
  xmlns="http://www.example.com/Report"
  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.example.com/Report  report.xsd"
  period="P3M" periodEnding="1999-12-31">
 <regions>
  <zip code="95819">
   <part number="872-AA" quantity="1"/>
   <part number="926-AA" quantity="1"/>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
   <part number="833-AA" quantity="1"/>
   <part number="455-BX" quantity="1"/>
  </zip>
  <zip code="63143">
   <part number="455-BX" quantity="4"/>
  </zip>
 </regions>
 <parts>
  <part number="872-AA">Lawnmower</part>
  <part number="926-AA">Baby Monitor</part>
  <part number="833-AA">Lapis Necklace</part>
  <part number="455-BX">Sturdy Shelves</part>
 </parts>
</purchaseReport>
```

Example 17-4 report.xsd

```
<schema targetNamespace="http://www.example.com/Report"
        xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:r="http://www.example.com/Report"
        elementFormDefault="qualified">
 <annotation>
  <documentation xml:lang="en">
   Report schema for Example.com
   Copyright 2000 Example.com. All rights reserved.
  </documentation>
 </annotation>
 <element name="purchaseReport">
  <complexType>
   <sequence>
    <element name="regions" type="r:RegionsType">
     <keyref name="dummy2" refer="r:pNumKey">
      <selector xpath="r:zip/r:part"/>
      <field xpath="@number"/>
     </keyref>
    </element>
    <element name="parts" type="r:PartsType"/>
   </sequence>
   <attribute name="period"       type="duration"/>
   <attribute name="periodEnding" type="date"/>
  </complexType>
  <unique name="dummy1">
   <selector xpath="r:regions/r:zip"/>
   <field xpath="@code"/>
  </unique>
  <key name="pNumKey">
   <selector xpath="r:parts/r:part"/>
   <field xpath="@number"/>
  </key>
 </element>
 <complexType name="RegionsType">
  <sequence>
   <element name="zip" maxOccurs="unbounded">
    <complexType>
     <sequence>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
      <element name="part" maxOccurs="unbounded">
       <complexType>
        <complexContent>
         <restriction base="anyType">
          <attribute name="number"   type="r:SKU"/>
          <attribute name="quantity" type="positiveInteger"/>
         </restriction>
        </complexContent>
       </complexType>
      </element>
     </sequence>
     <attribute name="code" type="positiveInteger"/>
    </complexType>
   </element>
  </sequence>
 </complexType>
 <simpleType name="SKU">
  <restriction base="string">
   <pattern value="\d{3}-[A-Z]{2}"/>
  </restriction>
 </simpleType>
 <complexType name="PartsType">
  <sequence>
   <element name="part" maxOccurs="unbounded">
    <complexType>
     <simpleContent>
      <extension base="string">
       <attribute name="number" type="r:SKU"/>
      </extension>
     </simpleContent>
    </complexType>
   </element>
  </sequence>
 </complexType>
</schema>
```

17.1.5 Differences Between XML Schemas and DTDs

The XML Schema language includes most of the capabilities of the DTD specification. An XML schema serves a similar purpose to a DTD, but is more flexible in specifying document constraints.

Table 17-1 compares some features between the two validation mechanisms.

Table 17-1 Feature Comparison Between XML Schema and DTD

Feature	XML Schema	DTD
Element nesting	X	X
Element occurrence constraints	X	X
Permitted attributes	X	X
Attribute types and default values	X	X
Written in XML	X	
Namespace support	X	

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 17-1 (Cont.) Feature Comparison Between XML Schema and DTD

Feature	XML Schema	DTD
Built-In data types	X	
User-Defined data types	X	
Include/Import	X	
Refinement (inheritance)	X	

These reasons are probably the most persuasive for choosing XML schema validation over DTD validation:

- The XML Schema language enables you to define rules for the *content* of elements and attributes. You achieve control over content by using data types. With XML Schema data types you can more easily perform actions such as:
  - Declare which elements are to contain which types of data, for example, positive integers in one element and years in another
  - Process data obtained from a database
  - Define restrictions on data, for example, a number between 10 and 20
  - Define data formats, for example, dates in the form MM-DD-YYYY
  - Convert data between different data types, for example, strings to dates
- Unlike DTD grammar, documents written in the XML Schema language are themselves written in XML. Thus, you can perform these actions:
  - Use your XML parser to parse your XML schema
  - Process your XML schema with the XML Document Object Model (DOM)
  - Transform your XML document with Extensible Stylesheet Language Transformation (XSLT)
  - Reuse your XML schemas in other XML schemas
  - Extend your XML schema by adding elements and attributes
  - Reference multiple XML schemas from the same document

17.2 Using the XML Schema Processor: Overview

The Oracle XML Schema processor is a SAX-based XML schema validator that you can use to validate instance documents against an XML schema. The processor supports both language example (LAX) and strict validation.

You can use the processor in these ways:

- Enable it in the XML parser
- Use it with a DOM tree to validate whole or part of an XML document
- Use it as a component in a processing pipeline (like a content handler)

You can configure the schema processor in different ways depending on your requirements. For example, you can:

- Use a fixed XML schema or automatically build a schema based on the `schemaLocation` attributes in an instance document.

Copyright © 1999, 2025, Oracle and/or its affiliates.

- Set `XMLError` and `entityResolver` to gain better control over the validation process.
- Determine how much of an instance document is to be validated. You can use any of the validation modes specified in Table 12-1. You can also designate a type of element as the root of validation.

17.2.1 Using the XML Schema Processor for Java: Basic Process

XDK packages that are important for applications that process XML schemas are described.

These are the important packages for applications that process XML schemas:

- `oracle.xml.parser.v2`, which provides APIs for XML parsing
- `oracle.xml.parser.schema`, which provides APIs for XML Schema processing

The most important classes in the `oracle.xml.parser.schema` package are described in Table 17-2. These form the core of most XML schema applications.

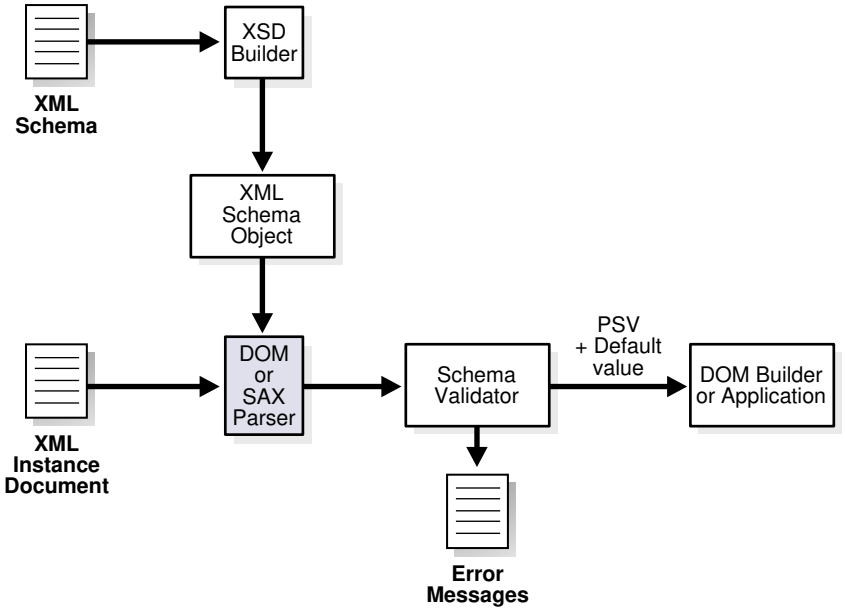
Table 17-2 oracle.xml.parser.schema Classes

Class/Interface	Description	Methods
`XMLSchema` class	Represents XML Schema component model. An `XMLSchema` object is a set of `XMLSchemaNodes` that belong to different target namespaces. The `XSDValidator` class uses `XMLSchema` for schema validation or metadata.	The principal methods are: • `get` methods such as `getElement()` and `getSchemaTargetNS()` get information about the XML schema • `printSchema()` prints information about the XML schema
`XMLSchemaNode` class	Represents schema components in a target namespace, including type definitions, element and attribute delcarations, and group and attribute group definitions.	The principal methods are `get` methods such as `getElementSet()` and `getAttributeDeclarations()` get components of the XML schema.
`XSDBuilder` class	Builds an `XMLSchema` object from an XML schema document. The `XMLSchema` object is a set of objects (Infoset items) corresponding to top-level schema declarations and definitions. The schema document is XML parsed and converted to a DOM tree.	The principal methods are: • `build()` creates an `XMLSchema` object. • `getObject()` returns the `XMLSchema` object. • `setEntityResolver()` sets an `EntityResolver` for resolving imports and includes.
`XSDValidator` class	Validates an instance XML document against an XML schema. When registered, an `XSDValidator` object is inserted as a pipeline node between `XMLParser` and `XMLDocument` events handlers.	The principal methods are: • `get` methods such as `getCurrentMode()` and `getElementDeclaration()` • `set` methods such as `setXMLProperty()` and `setDocumentLocator()` • `startDocument()` receives notification of the beginning of the document. • `startElement()` receives notification of the beginning of the element.

Figure 17-1 depicts the basic process of validating an instance document with the XML Schema processor for Java.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Figure 17-1 XML Schema Processor for Java



The XML Schema processor performs these major tasks:

1. A builder (`XSDBuilder` object) assembles the XML schema from an input XML schema document. Although instance documents and schemas need not exist specifically as files on the operating system, they are commonly referred to as files. They may exist as streams of bytes, fields in a database record, or collections of XML Infoset "Information Items."

   This task involves parsing the schema document into an object. The builder creates the schema object explicitly or implicitly:

   - In explicit mode, you pass in an XML schema when you invoke the processor. Validating Against Externally Referenced XML Schemas explains how to build the schema object in explicit mode.
   - In implicit mode, you do not pass in an XML schema when you invoke the processor because the schema is internally referenced by the instance document. Validating Against Internally Referenced XML Schemas explains how to create the schema object in implicit mode.

2. The XML schema validator uses the schema object to validate the instance document. This task has these steps:

   a. A Simple API for XML (SAX) parser parses the instance document into SAX events, which it passes to the validator.

   b. The validator receives SAX events as input and validates them against the schema object, sending an error message if it finds invalid XML components.

      Validation in the XML Parser describes the validation modes that you can use when validating the instance document. If you do not explicitly set a schema for validation with the `XSDBuilder` class, then the instance document must have the correct `xsi:schemaLocation` attribute pointing to the schema file. Otherwise, the program does not perform the validation. If the processor encounters errors, it generates error messages.

Copyright © 1999, 2025, Oracle and/or its affiliates.

c. The validator sends input SAX events, default values, or post-schema validation information to a DOM builder or application.

> **See Also**
>
> - *Oracle AI Database XML Java API Reference* to learn about the `XSDBuilder`, `DOMParser`, and `SAXParser` classes
> - Using the XML Schema Processor for Java to learn about the XDK SAX and DOM parsers

17.2.2 Running the XML Schema Processor Demo Programs

Demo programs for the XML Schema processor for Java are included in `$ORACLE_HOME/xdk/demo/java/schema`.

Table 17-3 describes the XML files and programs that you can use to test the XML Schema processor.

Table 17-3 XML Schema Sample Files

File	Description
`cat.xsd`	A sample XML schema used by the `XSDSetSchema.java` program to validate `catalogue.xml`. The `cat.xsd` schema specifies the structure of a catalogue of books.
`catalogue.xml`	A sample instance document that the `XSDSetSchema.java` program uses to validate against the `cat.xsd` schema.
`catalogue_e.xml`	A sample instance document used by the `XSDSample.java` program. When the program tries to validate this document against the `cat.xsd` schema, it generates schema errors.
`DTD2Schema.java`	This sample program converts a DTD (first argument) into an XML Schema and uses it to validate an XML file (second argument).
`embeded_xsql.xsd`	The XML schema used by `XSDLax.java`. The schema defines the structure of an XSQL page.
`embeded_xsql.xml`	The instance document used by `XSDLax.java`.
`juicer1.xml`	A sample XML document for use with `xsdproperty.java`. The XML schema that defines this document is `juicer1.xsd`.
`juicer1.xsd`	A sample XML schema for use with `xsdproperty.java`. This XML schema defines `juicer1.xml`.
`juicer2.xml`	A sample XML document for use with `xsdproperty.java`. The XML schema that defines this document is `juicer2.xsd`.
`juicer2.xsd`	A sample XML document for use with `xsdproperty.java`. This XML schema defines `juicer2.xml`.
`report.xml`	The sample XML file that `XSDSetSchema.java` uses to validate against the XML schema `report.xsd`.
`report.xsd`	A sample XML schema used by the `XSDSetSchema.java` program to validate the contents of `report.xml`. The `report.xsd` schema specifies the structure of a purchase order.
`report_e.xml`	When the program validates this sample XML file using `XSDSample.java`, it generates XML Schema errors.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 17-3 (Cont.) XML Schema Sample Files

File	Description
xsddom.java	This program shows how to validate an instance document by get a DOM representation of the document and using an XSDValidator object to validate it.
xsdent.java	This program validates an XML document by redirecting the referenced schema in the SchemaLocation attribute to a local version.
xsdent.xml	This XML document describes a book. The file is used as an input to xsdent.java.
xsdent.xsd	This XML schema document defines the rules for xsdent.xml. The schema document contains a schemaLocation attribute set to xsdent-1.xsd.
xsdent-1.xsd	The XML schema document referenced by the schemaLocation attribute in xsdent.xsd.
xsdproperty.java	This demo shows how to configure the XML Schema processor to validate an XML document based on a complex type or element declaration.
xsdsax.java	This demo shows how to validate an XML document received as a SAX stream.
XSDLax.java	This demo is the same as XSDSetSchema.java but sets the SCHEMA_LAX_VALIDATION flag for LAX validation.
XSDSample.java	This program is a sample driver that you can use to process XML instance documents.
XSDSetSchema.java	This program is a sample driver to process XML instance documents by overriding the schemaLocation. The program uses the XML Schema specification from cat.xsd to validate the contents of catalogue.xml.

Documentation for how to compile and run the sample programs is located in the README in the same directory. The basic steps are:

1. Change into the $ORACLE_HOME/xdk/demo/java/schema directory (UNIX) or %ORACLE_HOME%\xdk\demo\java\schema directory (Windows).
2. Run make (UNIX) or Make.bat (Windows) at the command line.
3. Add xmlparserv2.jar, xschema.jar, and the current directory to the CLASSPATH. These JAR files are located in $ORACLE_HOME/lib (UNIX) and %ORACLE_HOME%\lib (Windows). For example, you can set the CLASSPATH with the tcsh shell on UNIX:

   ```
   setenv CLASSPATH
    "$CLASSPATH":$ORACLE_HOME/lib/xmlparserv2.jar:$ORACLE_HOME/lib/schema.jar:.
   ```

   > **Note**
   >
   > The XML Schema processor requires JDK version 1.2 or later, and it is usable on any operating system with Java 1.2 support.

4. Run the sample programs with the XML files that are included in the directory:
   - These examples use report.xsd to validate the contents of report.xml:

     ```
     java XSDSample report.xml
     java XSDSetSchema report.xsd report.xml
     ```


Copyright © 1999, 2025, Oracle and/or its affiliates.

- This example validates an instance document in Lax mode:

```
java XSDLax embeded_xsql.xsd embeded_xsql.xml
```

- These examples use `cat.xsd` to validate the contents of `catalogue.xml`:

```
java XSDSample catalogue.xml
java XSDSetSchema cat.xsd catalogue.xml
```

- These examples generates error messages:

```
java XSDSample catalogue_e.xml
java XSDSample report_e.xml
```

- This example uses the `schemaLocation` attribute in `xsdent.xsd` to redirect the XML schema to `xsdent-1.xsd` for validation:

```
java xsdent xsdent.xml xsdent.xsd
```

- This example generates a SAX stream from `report.xml` and validates it against the XML schema defined in `report.xsd`:

```
java xsdsax report.xsd report.xml
```

- This example creates a DOM representation of `report.xml` and validates it against the XML schema defined in `report.xsd`:

```
java xsddom report.xsd report.xml
```

- These examples configure validation starting with an element declaration or complex type definition:

```
java xsdproperty juicer1.xml juicer1.xsd http://www.juicers.org \
juicersType false > juicersType.out


java xsdproperty juicer2.xml juicer2.xsd http://www.juicers.org \
Juicers true > juicers_e.out
```

- This example converts a DTD (`dtd2schema.dtd`) into an XML schema and uses it to validate an instance document (`dtd2schema.xml`):

```
java DTD2Schema dtd2schema.dtd dtd2schema.xml
```

17.2.3 Using the XML Schema Processor Command-Line Utility

You can use the XML parser command-line utility (`oraxml`) to validate instance documents against XML schemas and DTDs.

> **See Also**
>
> Using the Java XML Parser Command-Line Utility (oraxml) for information about how to run `oraxml` .

17.2.3.1 Using oraxml to Validate Against a Schema

An example shows how you can validate document `report.xml` against the XML schema `report.xsd` by invoking `oraxml` on the command line.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Example 17-5 Using oraxml to Validate Against a Schema

Invoke this command in directory `$ORACLE_HOME/xdk/demo/java/schema`:

```
oraxml -schema -enc report.xml
```

The expected output is:

```
The encoding of the input file: UTF-8
The input XML file is parsed without errors using Schema validation mode.
```

17.2.3.2 Using oraxml to Validate Against a DTD

An example shows how you can validate document `family.xml` against the DTD `family.dtd` by invoking `oraxml` on the command line.

Example 17-6 Using oraxml to Validate Against a DTD

Invoke this command in directory `$ORACLE_HOME/xdk/demo/java/parser/common`:

```
oraxml -dtd -enc family.xml
```

The expected output is:

```
The encoding of the input file: UTF-8
 The input XML file is parsed without errors using DTD validation mode.
```

17.3 Validating XML with XML Schemas

Topics cover various ways to validate XML documents using XML schemas.

17.3.1 Validating Against Internally Referenced XML Schemas

`$ORACLE_HOME/xdk/demo/java/schema/XSDSample.java` shows how to validate against an implicit XML Schema. The validation mode is implicit because the XML schema is referenced in the instance document itself.

Follow the steps in this section to write programs that use the `setValidationMode()` method of the `oracle.xml.parser.v2.DOMParser` class:

1. Create a DOM parser to use for the validation of an instance document. this code fragment from `XSDSample.java` shows how to create the `DOMParser` object:

```
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]);
   }
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
   public static void process (String xmlURI) throws Exception
   {
      DOMParser dp  = new DOMParser();
      URL       url = createURL(xmlURI);
      ...
   }
...
}
```

`createURL()` is a helper method that constructs a URL from a file name passed to the program as an argument.

2. Set the validation mode for the validating DOM parser with the `DOMParser.setValidationMode()` method. For example, `XSDSample.java` shows how to specify XML schema validation:

   ```
   dp.setValidationMode(XMLParser.SCHEMA_VALIDATION);
   dp.setPreserveWhitespace(true);
   ```

3. Set the output error stream with the `DOMParser.setErrorStream()` method. For example, `XSDSample.java` sets the error stream for the DOM parser object:

   ```
   dp.setErrorStream (System.out);
   ```

4. Validate the instance document with the `DOMParser.parse()` method. You do not have to create an XML schema object explicitly because the schema is internally referenced by the instance document. For example, `XSDSample.java` validates the instance document:

   ```
   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());
   }
   ```

17.3.2 Validating Against Externally Referenced XML Schemas

`$ORACLE_HOME/xdk/demo/java/schema/XSDSetSchema.java` shows how to validate an XML schema explicitly. The validation mode is explicit because you use the `XSDBuilder` class to specify the schema to use for validation: the schema is not specified in the instance document as in implicit validation.

Follow the basic steps in this section to write Java programs that use the `build()` method of the `oracle.xml.parser.schema.XSDBuilder` class:

1. Build an XML schema object from the XML schema document with the `XSDBuilder.build()` method. This code fragment from `XSDSetSchema.java` shows how to create the object:

   ```
   public class XSDSetSchema
   {
      public static void main(String[] args) throws Exception
   ```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
   {
      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);
   }
. . .
```

The `createURL()` method is a helper method that constructs a URL from the schema document file name specified on the command line.

2. Create a DOM parser to use for validation of the instance document. This code from `XSDSetSchema.java` shows how to pass the instance document file name and XML schema object to the `process()` method:

```
public static void process(String xmlURI, XMLSchema schemadoc)throws Exception{
   DOMParser dp  = new DOMParser();
   URL       url = createURL (xmlURI);
   . . .
```

3. Specify the schema object to use for validation with the `DOMParser.setXMLSchema()` method. This step is not necessary in implicit validation mode because the instance document already references the schema. For example, `XSDSetSchema.java` specifies the schema:

```
dp.setXMLSchema(schemadoc);
```

4. Set the validation mode for the DOM parser object with the `DOMParser.setValidationMode()` method. For example, `XSDSample.java` shows how to specify XML schema validation:

```
dp.setValidationMode(XMLParser.SCHEMA_VALIDATION);
dp.setPreserveWhitespace(true);
```

5. Set the output error stream for the parser with the `DOMParser.setErrorStream()` method. For example, `XSDSetSchema.java` sets it:

```
dp.setErrorStream (System.out);
```

6. Validate the instance document against the XML schema with the `DOMParser.parse()` method. For example, `XSDSetSchema.java` includes this code:

```
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());
}
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

17.3.3 Validating a Subsection of an XML Document

In LAX mode, you can validate parts of an XML document without validating all of it. LAX parsing validates elements in a document that are declared in an associated XML schema. The processor does not consider the instance document invalid if it contains no elements declared in the schema.

By using LAX mode, you can define the schema only for the part of the XML to be validated. The `$ORACLE_HOME/xdk/demo/java/schema/XSDLax.java` program shows how to use LAX validation. The program follows the basic steps described in Validating Against Externally Referenced XML Schemas:

1. Build an XML schema object from the user-specified XML schema document.
2. Create a DOM parser to use for validation of the instance document.
3. Specify the XML schema to use for validation.
4. Set the validation mode for the DOM parser object.
5. Set the output error stream for the parser.
6. Validate the instance document against the XML schema by invoking `DOMParser.parse()`.

To enable LAX validation, the program sets the validation mode in the parser to `SCHEMA_LAX_VALIDATION` rather than to `SCHEMA_VALIDATION`. This code fragment from `XSDLax.java` shows this technique:

```
dp.setXMLSchema(schemadoc);
dp.setValidationMode(XMLParser.SCHEMA_LAX_VALIDATION);
dp.setPreserveWhitespace (true);
. . .
```

You can test LAX validation by running the sample program:

```
java XSDLax embeded_xsql.xsd embeded_xsql.xml
```

17.3.4 Validating XML from a SAX Stream

`$ORACLE_HOME/xdk/demo/java/schema/xsdsax.java` shows how to validate an XML document received as a SAX stream. You instantiate an `XSDValidator` and register it with the SAX parser as the content handler.

Follow the steps in this section to write programs that validate XML from a SAX stream:

1. Build an XML schema object from the user-specified XML schema document by invoking the `XSDBuilder.build()` method. This code fragment shows how to create the object:

   ```
   XSDBuilder builder = new XSDBuilder();
   URL    url =  XMLUtil.createURL(args[0]);

   // Build XML Schema Object
   XMLSchema schemadoc = (XMLSchema)builder.build(url);
   process(args[1], schemadoc);
   . . .
   ```

   `createURL()` is a helper method that constructs a URL from the file name specified on the command line.

Copyright © 1999, 2025, Oracle and/or its affiliates.

2. Create a SAX parser (`SAXParser` object) to use for validation of the instance document. This code fragment from `saxxsd.java` passes the handles to the XML document and schema document to the `process()` method:

```
process(args[1], schemadoc);...public static void process(String xmlURI, XMLSchema
schemadoc)
throws Exception
{
    SAXParser dp  = new SAXParser();
...
```

3. Configure the SAX parser. This code fragment sets the validation mode for the SAX parser object with the `XSDBuilder.setValidationMode()` method:

```
dp.setPreserveWhitespace (true);
dp.setValidationMode(XMLParser.NONVALIDATING);
```

4. Create and configure a validator (`XSDValidator` object). This code fragment shows this technique:

```
XMLError err;... err = new XMLError();
...
XSDValidator validator = new XSDValidator();
...
validator.setError(err);
```

5. Specify the XML schema to use for validation by invoking the `XSDBuilder.setXMLProperty()` method. The first argument is the name of the property, which is `fixedSchema`, and the second is the reference to the XML schema object. This code fragment shows this technique:

```
validator.setXMLProperty(XSDNode.FIXED_SCHEMA, schemadoc);
...
```

6. Register the validator as the SAX content handler for the parser. This code fragment shows this technique:

```
dp.setContentHandler(validator);
...
```

7. Validate the instance document against the XML schema by invoking the `SAXParser.parse()` method. This code fragment shows this technique:

```
dp.parse (url);
```

17.3.5 Validating XML from a DOM

`$ORACLE_HOME/xdk/demo/java/schema/xsddom.java` shows how to validate an instance document by get a DOM representation of the document and using an `XSDValidator` object to validate it.

The `xsddom.java` program follows these steps:

1. Build an XML schema object from the user-specified XML schema document by invoking the `XSDBuilder.build()` method. This code fragment shows how to create the object:

```
XSDBuilder builder = new XSDBuilder();
URL    url =  XMLUtil.createURL(args[0]);

XMLSchema schemadoc = (XMLSchema)builder.build(url);
process(args[1], schemadoc);
```

`createURL()` is a helper method that constructs a URL from the file name specified on the command line.

Copyright © 1999, 2025, Oracle and/or its affiliates.

2. Create a DOM parser (`DOMParser` object) to use for validation of the instance document. This code fragment from `domxsd.java` passes the handles to the XML document and schema document to the `process()` method:

```
process(args[1], schemadoc);...public static void process(String xmlURI, XMLSchema
schemadoc)
throws Exception
{
    DOMParser dp  = new DOMParser();
    . . .
```

3. Configure the DOM parser. This code fragment sets the validation mode for the parser object with the `DOMParser.setValidationMode()` method:

```
dp.setPreserveWhitespace (true);
dp.setValidationMode(XMLParser.NONVALIDATING);
dp.setErrorStream (System.out);
```

4. Parse the instance document. This code fragment shows this technique:

```
dp.parse (url);
```

5. Get the DOM representation of the input document. This code fragment shows this technique:

```
XMLDocument doc = dp.getDocument();
```

6. Create and configure a validator (`XSDValidator` object). This code fragment shows this technique:

```
XMLError err;... err = new XMLError();
...
XSDValidator validator = new XSDValidator();
...
validator.setError(err);
```

7. Specify the schema object to use for validation by invoking the `XSDBuilder.setXMLProperty()` method. The first argument is the name of the property, which in this example is `fixedSchema`, and the second is the reference to the schema object. This code fragment shows this technique:

```
validator.setXMLProperty(XSDNode.FIXED_SCHEMA, schemadoc);
. . .
```

8. Get the root element (`XMLElement`) of the DOM tree and validate. This code fragment shows this technique:

```
XMLElement root = (XMLElement)doc.getDocumentElement();
XMLElement copy = (XMLElement)root.validateContent(validator, true);
copy.print(System.out);
```

17.3.6 Validating XML from Designed Types and Elements

`$ORACLE_HOME/xdk/demo/java/schema/xsdproperty.java` shows how to configure the XML Schema processor to validate an XML document based on a complex type or element declaration.

The `xsdproperty.java` program follows these steps:

1. Create `String` objects for the instance document name, XML schema name, root node namespace, root node local name, and specification of element or complex type ("true" means the root node is an element declaration). This code fragment shows this technique:

Copyright © 1999, 2025, Oracle and/or its affiliates.

```
String xmlfile = args[0];
String xsdfile =  args[1];
...
String ns = args[2]; //namespace for the root node
String nm = args[3]; //root node's local name
String el = args[4]; //true if root node is element declaration,
                     // otherwise, the root node is a complex type
```

2. Create an XSD builder and use it to create the schema object. This code fragment shows this technique:

```
XSDBuilder builder = new XSDBuilder();
URL    url =  XMLUtil.createURL(xsdfile);
XMLSchema  schema;
...
schema = (XMLSchema) builder.build(url);
```

3. Get the node. Invoke different methods depending on whether the node is an element declaration or a complex type:

   - If the node is an element declaration, pass the local name and namespace to the `getElement()` method of the schema object.
   - If the node is an element declaration, pass the namespace, local name, and root complex type to the `getType()` method of the schema object.

   `xsdproperty.java` uses this control structure:

```
QxName qname = new QxName(ns, nm);
...
XSDNode nd;
...
if (el.equals("true"))
{
  nd = schema.getElement(ns, nm);
  /* process ... */
}
else
{
  nd = schema.getType(ns, nm, XSDNode.TYPE);
  /* process ... */
}
```

4. After getting the node, create a new parser and set the schema to the parser to enable schema validation. This code fragment shows this technique:

```
DOMParser dp  = new DOMParser();
URL       url = XMLUtil.createURL (xmlURI);
```

5. Set properties on the parser and then parse the URL. Invoke the `schemaValidatorProperty()` method:

   a. Set the root element or type property on the parser to a fully qualified name.

      For a top-level element declaration, set the property name to `XSDNode.ROOT_ELEMENT` and the value to a `QName`, as showd by the `process1()` method.

      For a top-level type definition, set the property name to `XSDNode.ROOT_TYPE` and the value to a `QName`, as showd by the `process2()` method.

   b. Set the root node property on the parser to an element or complex type node.

      For an element node, set the property name to `XSDNode.ROOT_NODE` and the value to an `XSDElement` node, as showd by the `process3()` method.

Copyright © 1999, 2025, Oracle and/or its affiliates.

For a type node, set the property name to `XSDNode.ROOT_NODE` and the value to an `XSDComplexType` node, as showd by the `process3()` method.

This code fragment shows the sequence of method invocation:

```
if (el.equals("true"))
{
   nd = schema.getElement(ns, nm);
   process1(xmlfile, schema, qname);
   process3(xmlfile, schema, nd);
}
else
{
   nd = schema.getType(ns, nm, XSDNode.TYPE);
   process2(xmlfile, schema, qname);
   process3(xmlfile, schema, nd);
}
```

The processing methods are implemented:

```
  static void process1(String xmlURI, XMLSchema schema, QxName qname)
      throws Exception
  {
    /* create parser... */
    dp.setXMLSchema(schema);
    dp.setSchemaValidatorProperty(XSDNode.ROOT_ELEMENT, qname);
    dp.setPreserveWhitespace (true);
    dp.setErrorStream (System.out);
    dp.parse (url);
    ...
  }


  static void process2(String xmlURI, XMLSchema schema, QxName qname)
      throws Exception
  {
      /* create parser...
*/

    dp.setXMLSchema(schema);
    dp.setSchemaValidatorProperty(XSDNode.ROOT_TYPE, qname);
    dp.setPreserveWhitespace (true);
    dp.setErrorStream (System.out);
    dp.parse (url);
    ...
  }


  static void process3(String xmlURI, XMLSchema schema, XSDNode node)
      throws Exception
  {
      /* create parser... */


    dp.setXMLSchema(schema);
    dp.setSchemaValidatorProperty(XSDNode.ROOT_NODE, node);
    dp.setPreserveWhitespace (true);
    dp.setErrorStream (System.out);
    dp.parse (url);
    ...
  }
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

17.4 Tips and Techniques for Programming with XML Schemas

Topics include overriding schema location and converting a DTD to an XML schema.

17.4.1 Overriding the Schema Location with an Entity Resolver

When `XSDBuilder` builds a schema, it might need to include or import other schemas that are specified as URLs in a `schemaLocation` attribute. In some situations, you might want to override the schema locations specified in `<import>` and supply the builder with the required schema documents.

The `xsdent.java` demo described in Table 17-3 shows a case where a schema specified as `schemaLocation` needs to be imported. The document element in `xsdent.xml` file contains this attribute:

```
xsi:schemaLocation =  "http://www.example.com/BookCatalogue
                       xsdent.xsd">
```

The `xsdent.xsd` document contains these elements:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
               targetNamespace="http://www.example.com/BookCatalogue"
               xmlns:catd = "http://www.example.com/Digest"
               xmlns:cat  = "http://www.example.com/BookCatalogue"
               elementFormDefault="qualified">
<import namespace = "http://www.example.com/Digest"
        schemaLocation = "xsdent-1.xsd" />
```

As an example of wanting to override schema locations specified in `<import>` and supplying the builder with the required schema documents, suppose that you have downloaded the schemas documents from external web sites and stored them in a database. In such a situation, you can set an entity resolver in the `XSDBuilder`. `XSDBuilder` passes the schema location to the resolver, which returns an `InputStream`, `Reader`, or `URL` as an `InputSource`. The builder can read the schema documents from the `InputSource`.

The `xsdent.java` program shows how you can override the schema location with an entity resolver. You must implement the `EntityResolver` interface, instantiate the entity resolver, and set it in the XML schema builder. In the demo code, `sampleEntityResolver1` returns `InputSource` as an `InputStream` whereas `sampleEntityResolver2` returns `InputSource` as a `URL`.

Follow these basic steps:

1. Create a new XML schema builder:

   ```
   XSDBuilder builder = new XSDBuilder();
   ```

2. Set the builder to your entity resolver. An entity resolver is a class that implements the `EntityResolver` interface. The purpose of the resolver is to enable the XML reader to intercept any external entities before including them. This code fragment creates an entity resolver and sets it in the builder:

   ```
   builder.setEntityResolver(new sampleEntityResolver1());
   ```

   The `sampleEntityResolver1` class implements the `resolveEntity()` method. You can use this method to redirect external system identifiers to local URIs. The source code is:

Copyright © 1999, 2025, Oracle and/or its affiliates.

```
class sampleEntityResolver1 implements EntityResolver
{
   public InputSource resolveEntity (String targetNS,  String systemId)
   throws SAXException, IOException
   {
      // perform any validation check if needed based on targetNS & systemId
      InputSource mySource = null;
      URL u = XMLUtil.createURL(systemId);
      // Create input source with InputStream as input
      mySource = new InputSource(u.openStream());
      mySource.setSystemId(systemId);
      return mySource;
   }
}
```

The `sampleEntityResolver1` class initializes the `InputSource` with a stream.

3. Build the XML schema object. This code shows this technique:

```
schemadoc = builder.build(url);
```

4. Validate the instance document against the XML schema. The program executes this statement:

```
process(xmlfile, schemadoc);
```

The `process()` method creates a DOM parser, configures it, and invokes the `parse()` method. The method is implemented:

```
public static void process(String xmlURI, Object schemadoc)
    throws Exception
{
  DOMParser dp  = new DOMParser();
  URL       url = XMLUtil.createURL (xmlURI);

  dp.setXMLSchema(schemadoc);
  dp.setValidationMode(XMLParser.SCHEMA_VALIDATION);
  dp.setPreserveWhitespace (true);
  dp.setErrorStream (System.out);
  try {
     dp.parse (url);
     ...
}
```

17.4.2 Converting DTDs to XML Schemas

Because of the power and flexibility of the XML Schema language, you may want to convert your existing DTDs to XML schema documents. You can use XDK to perform this transformation.

The `$ORACLE_HOME/xdk/demo/java/schema/DTD2Schema.java` program shows how to convert a DTD. You can test the program:

```
java DTD2Schema dtd2schema.dtd dtd2schema.xml
```

Follow these basic steps to convert a DTD to an XML schema document:

1. Parse the DTD with the `DOMParser.parseDTD()` method. This code fragment from `DTD2Schema.java` shows how to create the DTD object:

```
XSDBuilder builder = new XSDBuilder();
URL dtdURL = createURL(args[0]);
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
DTD dtd = getDTD(dtdURL, "abc");
```

The `getDTD()` method is implemented:

```
private static DTD getDTD(URL dtdURL, String rootName)
   throws Exception
{
   DOMParser parser = new DOMParser();
   DTD dtd;
   parser.setValidationMode(true);
   parser.setErrorStream(System.out);
   parser.showWarnings(true);
   parser.parseDTD(dtdURL, rootName);
   dtd = (DTD)parser.getDoctype();
   return dtd;
}
```

2. Convert the DTD to an XML schema DOM tree with the `DTD.convertDTD2Sdhema()` method. This code fragment from `DTD2Schema.java` shows this technique:

   ```
   XMLDocument dtddoc = dtd.convertDTD2Schema();
   ```

3. Write the XML schema DOM tree to an output stream with the `XMLDocument.print()` method. This code fragment from `DTD2Schema.java` shows this technique:

   ```
   FileOutputStream fos = new FileOutputStream("dtd2schema.xsd.out");
   dtddoc.print(fos);
   ```

4. Create an XML schema object from the schema DOM tree with the `XSDBuilder.build()` method. This code fragment from `DTD2Schema.java` shows this technique:

   ```
   XMLSchema schemadoc = (XMLSchema)builder.build(dtddoc, null);
   ```

5. Validate an instance document against the XML schema with the `DOMParser.parse()` method. This code fragment from `DTD2Schema.java` shows this technique:

   ```
   validate(args[1], schemadoc);
   ```

   The `validate()` method is implemented:

   ```
   DOMParser dp  = new DOMParser();
   URL       url = createURL (xmlURI);
   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");
   }
   ...
   ```


Copyright © 1999, 2025, Oracle and/or its affiliates.

18

Using the JAXB Class Generator

An explanation is given of how to use the Java Architecture for XML Binding (JAXB) class generator.

> **Note**
>
> Use the Java Architecture for XML Binding (JAXB) class generator for new applications to take advantage of the object binding feature for Extensible Markup Language (XML) data. The Oracle9i class generator for Java is deprecated. Oracle Database 10g supports the Oracle9i class generator for backward compatibility.

18.1 Introduction to the JAXB Class Generator

Topics introducing the JAXB class generator include prerequisites, standards and specifications, marshalling and unmarshalling, validation, and customization.

18.1.1 Prerequisites for Using the JAXB Class Generator

Prerequisites for using the JAXB class generator are listed.

This chapter assumes that you have some familiarity with these topics:

- Java Architecture for XML Binding (JAXB). For a more thorough introduction to JAXB than is possible in this chapter, consult the XML resources listed in Related Documents.
- XML Schema language. See Using the XML Schema Processor for Java for an overview and links to suggested reading.

18.1.2 Standards and Specifications for the JAXB Class Generator

The Oracle JAXB processor implements JSR-31, *The Java Architecture for XML Binding (JAXB)*, Version 1.0, which is a recommendation of the Java Community Process (JCP).

The Oracle XML Developer's Kit (XDK) implementation of the JAXB 1.0 specification does *not* support these *optional* features:

- Javadoc generation
- Fail Fast validation
- External customization file
- XML Schema concepts described in section E.2 of the specification

Related Topics

- Oracle XML Developer's Kit Standards
  A description is given of the Oracle XML Developer's Kit (XDK) standards.

Copyright © 1999, 2025, Oracle and/or its affiliates.

18.1.3 JAXB Class Generator Features

The JAXB class generator for Java generates the interfaces and the implementation classes corresponding to an XML Schema. Its principal advantage to Java developers is automation of the mapping between XML documents and Java code, which enables programs to use generated code to read, manipulate, and re-create XML data.

The Java classes, which can be extended, give the developer access to the XML data without knowledge of the underlying XML data structure.

The Oracle JAXB class generator provides these advantages for XML application development in Java:

- Speed

  Because the schema-to-code conversion is automated, you can rapidly generate Java code from an input XML schema.

- Ease of use

  You can invoke generated `get` and `set` methods rather than code your own from the start.

- Automated data conversion

  You can automate the conversion of XML document data into Java data types.

- Customization

  JAXB provides a flexible framework that enables you to customize the binding of XML elements and attributes.

18.1.4 Marshalling and Unmarshalling with JAXB

JAXB is an application programming interface (API) and set of tools that maps XML data to Java objects. JAXB simplifies access to an XML document from a Java program by presenting the XML document to the program in a Java format.

You can use the JAXB API and tools to perform these basic tasks:

1. Generate and compile JAXB classes from an XML schema with the `orajaxb` command-line utility.

   To use the JAXB class generator to generate Java classes you must provide it with an XML schema. Document type definitions (DTDs) are not supported by JAXB. As explained in Converting DTDs to XML Schemas, however, you can use the `DTD2Schema` program to convert a DTD to an XML schema. Afterwards, you can use the JAXB class generator to generate classes from the schema.

   The JAXB compiler generates Java classes that map to constraints in the source XML schema. The classes implements `get` and `set` methods that you can use to get and specify data for each type of element and attribute in the schema.

2. Process XML documents by instantiating the generated classes in a Java program.

   Specifically, you can write a program that uses the JAXB binding framework to perform these tasks:

   a. Unmarshal the XML documents.

      As explained in the JAXB specification, unmarshalling is defined as moving data from an XML document to the Java-generated objects.

   b. Validate the XML documents.

Copyright © 1999, 2025, Oracle and/or its affiliates.

You can validate before or during the unmarshalling of the contents into the content tree. You can also validate on demand by invoking the validation API on the Java object. See Validation with JAXB.

c. Modify Java content objects.

   The content tree of data objects represents the structure and content of the source XML documents. You can use the `set` methods defined for a class to modify the content of elements and attributes.

d. Marshal Java content objects back to XML.

   In contrast to unmarshalling, marshalling is creating an XML document from Java objects by traversing a content tree of instances of Java classes. You can serialize the data to a Document Object Model (DOM) tree, Simple API for XML (SAX) content handler, transformation result, or output stream.

18.1.5 Validation with JAXB

A Java content tree is considered valid with an XML schema when marshalling the tree generates a valid XML document.

JAXB applications can perform validation in these circumstances:

- Unmarshalling-time validation that notifies the application of errors and warnings during unmarshalling. If unmarshalling includes validation that is error-free, then the input XML document and the Java content tree are valid.
- On-demand validation of a Java content tree initiated by the application.
- Fail-fast validation that gives immediate results while updating the Java content tree with `set` and `get` methods. As specified in Standards and Specifications for the JAXB Class Generator, fail-fast validation is an optional feature in the JAXB 1.0 specification that is not supported in the XDK implementation of the JAXB class generator.

JAXB applications must be able to marshal a valid Java content tree, but they are not required to ensure that the Java content tree is valid before invoking a marshalling API. The marshalling process does not itself validate the content tree. Programs are required to throw a `javax/xml/bind/MarshalException` when marshalling fails due to invalid content.

18.1.6 JAXB Customization

The declared element and type names in an XML schema do not always provide the most useful Java class names. You can override the default JAXB bindings by using custom binding declarations, which are described in the JAXB specification.

These declarations let you customize your generated JAXB classes beyond the XML-specific constraints in an XML schema, to include Java-specific refinements such as class and package name mappings.

You can annotate the schema to perform these customizations:

- Bind XML names to user-defined Java class names
- Name the package, derived classes, and methods
- Choose which elements to bind to which classes
- Decide how to bind each attribute and element declaration to a property in the appropriate content class
- Choose the type of each attribute-value or content specification

Copyright © 1999, 2025, Oracle and/or its affiliates.

Several of the demos programs listed in Table 18-2 show JAXB customizations.

> **See Also**
>
> Customizing a Class Name in a Top-Level Element for a detailed explanation of a customization demo

18.2 Using the JAXB Class Generator: Overview

Topics here include the basic process of using the JAXB processor, running the XML schema processor demo programs, and using the JAXB class generator command-line utility.

18.2.1 Using the JAXB Processor: Basic Process

The XDK JAXB API basic process is described.

The XDK JAXB API exposes these packages:

- `javax.xml.bind`, which provides a runtime binding framework for client applications including unmarshalling, marshalling, and validation
- `javax.xml.bind.util`, which provides useful client utility classes

The most important classes and interfaces in the `javax.xml.bind` package are described in Table 18-1. These form the core of most JAXB applications.

Table 18-1 javax.xml.bind Classes and Interfaces

Class/Interface	Description	Methods
`JAXBContext` class	Provides an abstraction for managing the XML/Java binding information necessary to implement the JAXB binding framework operations: unmarshal, marshal, and validate. A client application gets new instances of this class by invoking the `newInstance()` method.	The principal methods are: • `newInstance()` creates a JAXB content class. Supply this method the name of the package containing the generated classes. • `createMarshaller()` creates a marshaller that you can use to convert a content tree to XML. • `createUnmarshaller()` creates an unmarshaller that you can use to convert XML to a content tree. • `createValidator()` creates a `Validator` object that can validate a java content tree against its source schema.
`Marshaller` interface	Governs the process of serializing Java content trees into XML data.	The principal methods are: • `getEventHandler()` returns the current or default event handler. • `getProperty()` gets the property in the underlying implementation of marshaller. • `marshal()` marshals the content tree into a DOM, SAX2 events, output stream, transformation result, or `Writer`. • `setEventHandler()` creates a `Validator` object that validates a java content tree against its source schema.

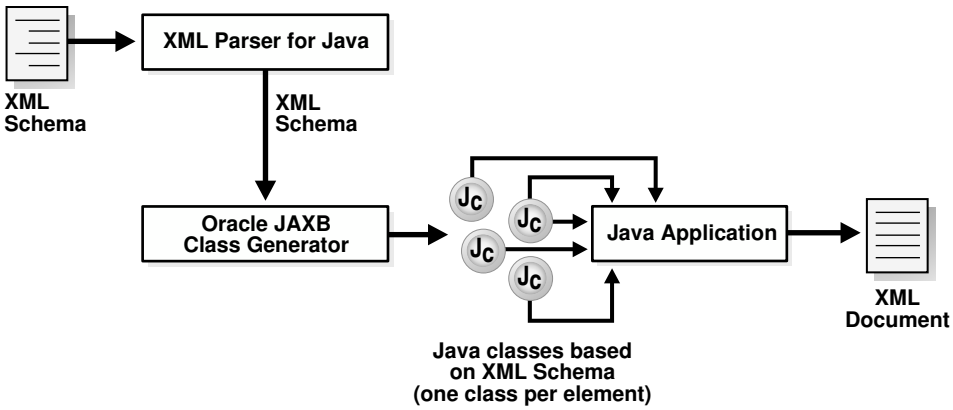
Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 18-1 (Cont.) javax.xml.bind Classes and Interfaces

Class/Interface	Description	Methods
`Unmarshaller` interface	Governs the process of deserializing XML data into newly created Java content trees, optionally validating the XML data as it is unmarshalled.	The principal methods are: • `getEventHandler()` returns the current or default event handler. • `getUnmarshallerHandler()` returns an unmarshaller handler object usable as a component in an XML pipeline. • `isValidating()` indicates whether the unmarshaller is set to validate mode. • `setEventHandler()` allows an application to register a `ValidationEventHandler`. • `setValidating()` specifies whether the unmarshaller validates during unmarshal operations. • `marshal()` unmarshals XML data from the specified file, URL, input stream, input source, SAX, or DOM.
`Validator` interface	Controls the validation of content trees during run time. Specifically, this interface controls on-demand validation, which enables clients to receive data about validation errors and warnings detected in the Java content tree.	The principal methods are: • `getEventHandler()` returns the current or default event handler. • `setEventHandler()` allows an application to register a `ValidationEventHandler`. • `validate()` validates Java content trees on-demand at run time. This method can validate any arbitrary subtree of the Java content tree. • `validateRoot()` validates the Java content tree rooted at `rootObj`. You can use this method to validate an entire Java content tree.

Figure 18-1 depicts the process flow of a framework that uses the JAXB class generator.

Figure 18-1 JAXB Class Generator for Java



The basic stages of the process shown in Figure 18-1 are:

1. The XML parser parses the XML schema and sends the parsed data to the JAXB class generator.

Copyright © 1999, 2025, Oracle and/or its affiliates.

2. The class generator creates Java classes and interfaces based on the input XML schema.

   By default, one XML element or type declaration generates one interface and one class. For example, if the schema defines an element named `<anElement>`, then by default the JAXB class generator generates a source file named `AnElement.java` and another named `AnElementImpl.java`. You can use customize binding declarations to override the default binding of XML Schema components to Java representations.

3. The Java compiler compiles the `.java` source files into class files. All of the generated classes, source files, and application code must be compiled.

4. Your Java application uses the compiled classes and the binding framework to perform these types of tasks:

   - Create a JAXB context. You use this context to create the marshaller and unmarshaller.
   - Build object trees representing XML data that is valid against the XML schema. You can perform this task by either unmarshalling the data from an XML document that conforms to the schema or instantiating the classes.
   - Access and modify the data.
   - Optionally validate the modifications to the data relative to the constraints expressed in the XML schema.
   - Marshal the data to new XML documents.

See Also

- *Oracle AI Database XML Java API Reference* for details of the JAXB API
- Processing XML with the JAXB Class Generator for detailed explanations of JAXB processing

18.2.2 Running the XML Schema Processor Demo Programs

Demo programs for the JAXB class generator for Java are included in `$ORACLE_HOME/xdk/demo/java/jaxb`.

Specifically, XDK includes the JAXB demos listed in Table 18-2.

Table 18-2 JAXB Class Generator Demos

Program	Subdirectory within Oracle Home	Demonstrates . . .
`SampleApp1.java`	`/xdk/demo/java/jaxb/Sample1`	The binding of top-level element and `complexType` definitions in the `sample1.xsd` schema to Java classes.
`SampleApp2.java`	`/xdk/demo/java/jaxb/Sample2`	The binding of a top-level element with an inline `simpleType` definition in the `sample2.xsd` schema.
`SampleApp3.java`	`/xdk/demo/java/jaxb/Sample3`	The binding of a top-level `complexType` element that is derived by extension from another top-level `complexType` definition. See Binding Complex Types for a detailed explanation of this program.
`SampleApp4.java`	`/xdk/demo/java/jaxb/Sample4`	The binding of a content model within a `complexType` that refers to a top-level named group.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 18-2 (Cont.) JAXB Class Generator Demos

Program	Subdirectory within Oracle Home	Demonstrates . . .
SampleApp5.java	/xdk/demo/java/jaxb/Sample5	The binding of <choice> with maxOccurs unbounded within a complexType.
SampleApp6.java	/xdk/demo/java/jaxb/Sample6	The binding of atomic data types.
SampleApp7.java	/xdk/demo/java/jaxb/Sample7	The binding a complexType definition in which mixed="true".
SampleApp8.java	/xdk/demo/java/jaxb/Sample8	The binding of elements and types declared in two different namespaces.
SampleApp9.java	/xdk/demo/java/jaxb/Sample9	The customization of a Java package name.
SampleApp10.java	/xdk/demo/java/jaxb/Sample10	The customization of class name in a top-level element. See Customizing a Class Name in a Top-Level Element for a detailed explanation of this program.
SampleApp11.java	/xdk/demo/java/jaxb/Sample11	The customization of class name of a local element occurring in a repeating model group declared inside a complexType element.
SampleApp12.java	/xdk/demo/java/jaxb/Sample12	The customization of the attribute name.
SampleApp13.java	/xdk/demo/java/jaxb/Sample13	The javaType customization specified on a global simpleType. The javaType customization specifies the parse and print method declared on a user-defined class.
SampleApp14.java	/xdk/demo/java/jaxb/Sample14	The customization of the typesafe enum class name.

You can find documentation that describes how to compile and run the sample programs in the README in the same directory. The basic steps are:

1. Change into the $ORACLE_HOME/xdk/demo/java/jaxb directory (UNIX) or %ORACLE_HOME%\xdk\demo\java\jaxb directory (Windows).
2. Make sure that your environment variables are set as described in Setting Up the XDK for Java Environment.
3. Run make (UNIX) or Make.bat (Windows) at the system prompt. The make utility performs these sequential actions for each sample subdirectory:
   a. Runs the orajaxb utility to generate Java class files based on an input XML schema. For most of the demos, the output classfiles are written to the generated subdirectory. For example, the make file performs these commands for the sample1.xsd schema in the Sample1 subdirectory:

      ```
      cd ./Sample1; $(JAVA_HOME)/bin/java -classpath "$(MAKE_CLASSPATH)" \
      oracle.xml.jaxb.orajaxb -schema sample1.xsd -targetPkg generated; echo;
      ```

   b. Runs the javac utility to compile the Java classes. For example, the make utility performs these commands for the Java class files in the Sample1/generated/ subdirectory:

      ```
      cd ./Sample1/generated; $(JAVA_HOME)/bin/javac -classpath \
      "$(MAKE_CLASSPATH)" *.java
      ```

   c. Runs the javac utility to compile a sample Java application that uses the classes compiled in the preceding step. For example, the make utility compiles the SampleApp1.java program:

      ```
      cd ./Sample1; $(JAVA_HOME)/bin/javac -classpath "$(MAKE_CLASSPATH)" \
      SampleApp1.java
      ```


Copyright © 1999, 2025, Oracle and/or its affiliates.

d. Runs the sample Java application and writes the results to a log file. For example, the `make` utility executes the `SampleApp1` class and writes the output to `sample1.out`:

```
cd ./Sample1; $(JAVA_HOME)/bin/java -classpath "$(MAKE_CLASSPATH)" \SampleApp1 >
sample1.out
```

18.2.3 Using the JAXB Class Generator Command-Line Utility

XDK includes `orajaxb`, which is a command-line Java interface that generates Java classes from input XML schemas. Shell scripts `$ORACLE_HOME/bin/orajaxb` and `%ORACLE_HOME%\bin\orajaxb.bat` execute class `oracle.xml.jaxb.orajaxb`.

To use `orajaxb` ensure that your `CLASSPATH` is set as described in Setting Up the XDK for Java Environment.

Table 18-3 lists the `orajaxb` command-line options.

Table 18-3 orajaxb Command-Line Options

Option	Purpose
`-help`	Prints the help message.
`-version`	Prints the release version.
`-outputdir` *`OutputDir`*	Specifies the directory in which to generate the Java source files. If the schema has a namespace, then the program generates the java code in the package (corresponding to the namespace) referenced from the outputDir. By default, the current directory is the *`outputDir`*.
`-schema` *`SchemaFile`*	Specifies the input XML schema.
`-targetPkg` *`targetPkg`*	Specifies the target package name. This option overrides any binding customization for package name, and also the default package name algorithm defined in the JAXB specification.
`-interface`	Generates only the interfaces.
`-verbose`	Lists the generated classes and interfaces.
`-defaultCus` *`fileName`*	Generates the default customization file.
`-extension`	Allows vendor specific extensions and does not strictly follow the compatibility rules specified in Appendix E.2 of the JAXB 1.0 specification. When specified, the program ignores JAXB 1.0 unsupported features such as notations, substitution groups, and any attributes.

18.2.3.1 Using the JAXB Class Generator Command-Line Utility: Example

An example shows how to use the JAXB class generator command-line utility.

To test `orjaxb`, change to directory `$ORACLE_HOME/xdk/demo/java/jaxb/Sample1`. If you have run `make` then the directory contains these files:

```
SampleApp1.class
SampleApp1.java
generated/
sample1.out
sample1.xml
sample1.xsd
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

File `sample.xsd` is the XML schema associated with XML document `sample1.xml`. Subdirectory `generated/` contains the classes generated from the input schema. You can test `orajaxb` by deleting the contents of `generated/` and regenerating the classes.

```
rm generated/*
orajaxb -schema sample1.xsd -targetPkg generated -verbose
```

The terminal displays this output:

```
generated/CType.java
generated/AComplexType.java
generated/AnElement.java
generated/RElemOfCTypeInSameNs.java
generated/RType.java
generated/RElemOfSTypeInSameNs.java

generated/CTypeImpl.java
generated/AComplexTypeImpl.java
generated/AnElementImpl.java
generated/RElemOfCTypeInSameNsImpl.java
generated/RTypeImpl.java
generated/RElemOfSTypeInSameNsImpl.java
generated/ObjectFactory.java
```

18.2.4 JAXB Features Not Supported in XDK

Features not supported by the XDK implementation of the JAXB specification are described.

The XDK implementation of the JAXB specification does not support these features:

- Javadoc generation
- XML Schema component "any" and substitution groups

18.3 Processing XML with the JAXB Class Generator

Topics include binding complex types and customizing a class name in a top-level element.

18.3.1 Binding Complex Types

`Sample3.java` shows how to bind a complex type definition to a Java content interface. One complex type defined in the XML schema is derived by extension from another complex type.

18.3.1.1 Defining the Schema to Validate sample3.xml

An XML schema, `schema3.xsd`, is defined for validating XML document `schema3.xml`.

Example 18-1 shows the XML data document that provides the input to the sample application. The `sample3.xml` document describes the address of an employee.

The XML schema shown in Example 18-2 defines the structure that you use to validate `sample3.xml`. The schema defines two complex types and one element, which are defined:

Copyright © 1999, 2025, Oracle and/or its affiliates.

- The first complex type, which is named `Address`, is a sequence of elements. Each element in the sequence describes one part of the address: name, door number, and so forth.
- The second complex type, which is named `USAddress`, uses the `<extension base="exp:Address">` element to extend `Address` by adding US-specific elements to the `Address` sequence: state, zip, and so forth. The `exp` prefix specifies the `http://www.oracle.com/sample3/` namespace.
- The element is named `myAddress` and is of type `exp:USAddress`. The `exp` prefix specifies the `http://www.oracle.com/sample3/` namespace. In `sample3.xml`, the `myAddress` top-level element, which is in namespace `http://www.oracle.com/sample3/`, conforms to the schema definition.

Example 18-1 sample3.xml

```
<?xml version="1.0"?>
<myAddress xmlns = "http://www.oracle.com/sample3/"
           xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation = "http://www.oracle.com/sample3 sample3.xsd">
    <name>James Bond</name>
    <doorNumber>420</doorNumber>
    <street>Oracle parkway</street>
    <city>Redwood shores</city>
    <state>CA</state>
    <zip>94065</zip>
    <country>United States</country>
</myAddress>
```

Example 18-2 sample3.xsd

```
<?xml version="1.0"?>

<!-- Binding a complex type definition to java content interface
 The complex type definition is derived by extension
-->

<schema xmlns = "http://www.w3.org/2001/XMLSchema"
        xmlns:exp="http://www.oracle.com/sample3/"
        targetNamespace="http://www.oracle.com/sample3/"
        elementFormDefault="qualified">

   <complexType name="Address">
      <sequence>
         <element name="name" type="string"/>
         <element name="doorNumber" type="short"/>
         <element name="street" type="string"/>
         <element name="city" type="string"/>
      </sequence>
   </complexType>

  <complexType name="USAddress">
    <complexContent>
     <extension base="exp:Address">
       <sequence>
          <element name="state" type="string"/>
          <element name="zip" type="integer"/>
          <element name="country" type="string"/>
       </sequence>
     </extension>
    </complexContent>
  </complexType>

  <element name="myAddress" type="exp:USAddress"/>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
</schema>
```

18.3.1.2 Generating and Compiling the Java Classes

If you have an XML document and corresponding XML schema, then the next stage of processing is to generate the Java classes from the XML schema.

You can use the JAXB command-line interface described in Using the JAXB Class Generator Command-Line Utility to perform this task.

Assuming that your environment is set up as described in Setting Up the XDK for Java Environment, you can create the source files in the `generated` package:

```
cd $ORACLE_HOME/xdk/demo/java/jaxb/Sample3
orajaxb -schema sample1.xsd -targetPkg generated
```

The preceding `orajaxb` command creates these source files in the `./generated/` subdirectory:

```
Address.java
AddressImpl.java
MyAddress.java
MyAddressImpl.java
ObjectFactory.java
USAddress.java
USAddressImpl.java
```

The complex types `Address` and `USAddress` each has two associated source files, as does the element `MyAddress`. The source file named after the element contains the interface; the file with the suffix `Impl` contains the class that implements the interface. For example, `Address.java` contains the interface `Address`, whereas `AddressImpl.java` contains the class that implements `Address`.

The content of the `Address.java` source file is shown in Example 18-3.

The `Address` complex type defined a sequence of elements: `name`, `doorNumber`, `street`, and `city`. Consequently, the `Address` interface includes a `get` and `set` method signature for each of the four elements. For example, the interface includes `getName()` for retrieving data in the `<name>` element and `setName()` for modifying data in this element.

You can compile the Java source files with `javac`:

```
cd $ORACLE_HOME/xdk/demo/java/jaxb/Sample3/generated
javac *.java
```

Example 18-3 Address.java

```
package generated;
public interface Address
{
   public void setName(java.lang.String n);
   public java.lang.String getName();
   public void setDoorNumber(short d);
   public short getDoorNumber();
   public void setStreet(java.lang.String s);
   public java.lang.String getStreet();
   public void setCity(java.lang.String c);
   public java.lang.String getCity();
}
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

18.3.1.3 Processing the XML Data in sample3.xml

`Sample3.java` unmarshals an XML data document, marshals it, and uses the generated classes to print and modify the address data.

It shows how you can process the `sample3.xml` document by using the Java class files that you generated in Generating and Compiling the Java Classes.

The `Sample3.java` program processes the data as follows:

1. Create strings for the XML data document file name and the name of the directory that contains the generated classes. This name is the package name. For example:

   ```
   String fileName = "sample3.xml";
   String instancePath = "generated";
   ```

2. Instantiate a JAXB context by invoking `JAXBContext.newInstance()`. A client application gets a new instance of this class by initializing it with a context path. The path contains a list of Java package names that contain the interfaces available to the marshaller. Thisthese statement shows this technique:

   ```
   JAXBContext jc = JAXBContext.newInstance(instancePath);
   ```

3. Instantiate the unmarshaller. The `Unmarshaller` class governs the process of deserializing XML data into newly created objects, optionally validating the XML data as it is unmarshalled. Thisthese statement shows this technique:

   ```
   Unmarshaller u = jc.createUnmarshaller();
   ```

4. Unmarshal the XML document. Invoke the `Unmarshaller.unmarshal()` method to deserialize the `sample3.xml` document and return the content trees as an `Object`. You can create a URL from the XML file name by invoking the `fileToUrl()` helper method. This statement shows the technique:

   ```
   Object obj = u.unmarshal(fileToURL(fileName));
   ```

5. Instantiate a marshaller. The `Marshaller` class governs the process of serializing Java content trees back into XML data. Thisthese statement shows this technique:

   ```
   Marshaller m = jc.createMarshaller();
   ```

6. Marshal the content tree. Invoke the `Marshaller.marshal()` method to marshal the content tree `Object` returned by the unmarshaller. You can serialize the data to a DOM tree, SAX content handler, transformation result, or output stream. This statement serializes the XML data, including markup, as an output stream:

   ```
   m.marshal(obj, System.out);
   ```

   By default, the marshaller uses 8-bit encoding of Unicode (UTF-8) encoding when writing XML data to an output stream.

7. Print the contents of the XML document. The program implements a `process()` method that accepts the content tree and marshaller as parameters.

   The first stage of processing prints the data in the XML document without the XML markup. The method casts the `Object` generated by the marshaller into type `MyAddress`. It proceeds to invoke a series of methods whose method names are constructed by prefixing `get` to the name of an XML element. For example, to get the data in the `<city>` element in Example 18-1, the program invokes `getCity()`. This code fragment shows this technique:

   ```
   public static void process(Object obj, Marshaller m) throws Throwable
   {
      generated.MyAddress elem = (generated.MyAddress)obj;
   ```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
        System.out.println();
        System.out.println(" My address is: ");
        System.out.println("  name:  "  + elem.getName() + "\n"  +
                           "  doorNumber " + elem.getDoorNumber() + "\n" +
                           "  street: " + elem.getStreet() + "\n" +
                           "  city:   " + elem.getCity() + "\n"  +
                           "  state:  " + elem.getState() + "\n" +
                           "  zip:  " + elem.getZip() + "\n" +
                           "  country:  " + elem.getCountry() + "\n" +
                           "\n" );
    ...
```

8. Change the XML data and print it. The `process()` method continues by invoking set methods that are analogous to the preceding get methods. The name of each set method is constructed by prefixing `set` to the name of an XML element. For example, `setCountry()` changes the value in the `<country>` element. These statements show this technique:

   ```
   short num = 550;
   elem.setDoorNumber(num);
   elem.setCountry("India");
   num = 10100;
   elem.setZip(new java.math.BigInteger("100100"));
   elem.setCity("Noida");
   elem.setState("Delhi");
   ```

   After changing the data, the program prints the data by invoking the same get methods as in the previous step.

18.3.2 Customizing a Class Name in a Top-Level Element

The `Sample10.java` program shows one form of JAXB customization. The program shows you can change the name of a class that corresponds to an element in the input XML schema.

18.3.2.1 Defining the Schema to Validate schema10.xml

An XML schema, `schema10.xsd`, is defined for validating XML document `schema10.xml`.

Example 18-4 shows the XML data document that provides the input to the sample application. The `sample10.xml` document describes a business.

Example 18-5 shows the XML schema that defines the structure of `sample10.xml`. The schema defines one complex type and one element as follows:

- The complex type, which is named `businessType`, is a sequence of elements. Each element in the sequence describes a part of the business: title, owner, and id.
- The element, which is named `business`, is of type `biz:businessType`. The `biz` prefix specifies the `http://jaxbcustomized/sample10/` namespace. In `sample10.xml`, the `business` top-level element, which is in namespace `http://jaxbcustomized/sample10/`, conforms to the schema definition.

Example 18-4 sample10.xml

```
<?xml version="1.0"?>
<business xmlns="http://jaxbcustomized/sample10/">
   <title>Software Development</title>
   <owner>Larry Peterson</owner>
   <id>45123</id>
</business>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

Example 18-5 sample10.xsd

```
<?xml version="1.0"?>

<!-- Customization of class name in top level element -->

<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://jaxbcustomized/sample10/"
        xmlns:biz="http://jaxbcustomized/sample10/"
        xmlns:jaxb="http://java.sun.com/xml/ns/jaxb"
        jaxb:version="1.0"
        elementFormDefault="qualified">

   <element name="business" type="biz:businessType">
      <annotation>
         <appinfo>
            <jaxb:class name="myBusiness"/>
         </appinfo>
      </annotation>
   </element>

   <complexType name="businessType">
      <sequence>
         <element name="title" type="string"/>
         <element name="owner" type="string"/>
         <element name="id" type="integer"/>
      </sequence>
   </complexType>

</schema>
```

18.3.2.1.1 Customizing the Schema Binding

Binding customizations used in XML schema `sample10.xsd` are described.

The schema shown in Example 18-5 customizes the binding of the `business` element with an inline binding declaration. The general form for inline customizations is:

```
<xs:annotation>
   <xs:appinfo>
      .
      .
      binding declarations
      .
      .
   </xs:appinfo>
</xs:annotation>
```

Example 18-5 uses the `<class>` binding declaration to bind a schema element to a Java class name. You can use the declaration to customize the name for an interface or the class that implements an interface. The JAXB class generator supports this syntax for `<class>` customizations:

```
<class [ name = "className"] >
```

The `name` attribute specifies the name of the derived Java interface. Example 18-5 contains this customization:

```
<jaxb:class name="myBusiness"/>
```

Thus, the schema binds the `business` element to the interface `myBusiness` rather than to the interface `business`, which is the default.

Copyright © 1999, 2025, Oracle and/or its affiliates.

18.3.2.2 Generating and Compiling the Java Classes

After you have an XML document and corresponding XML schema, the next stage is to generate the Java classes from the XML schema. You can use the JAXB command-line interface to perform this task.

If your environment is set up as described in Setting Up the XDK for Java Environment, then you can create the source files in the `generated` package:

```
cd $ORACLE_HOME/xdk/demo/java/jaxb/Sample10
orajaxb -schema sample10.xsd
```

Because the preceding command does not specify a target package, the package name is constructed from the target namespace of the schema, which is `http://jaxbcustomized/sample10/` . Consequently, the utility generates these source files in the `./jaxbcustomized/sample10/` subdirectory:

```
BusinessType.java
BusinessTypeImpl.java
MyBusiness.java
MyBusinessImpl.java
ObjectFactory.java
```

The complex type `businessType` has two source files, `BusinessType.java` and `BusinessTypeImpl.java`. Because of the JAXB customization, the `business` element is bound to interface `MyBusiness` and implementing class `MyBusinessImpl`.

The content of the `BusinessType.java` source file is shown in Example 18-6.

The `BusinessType` complex type defined a sequence of elements: `title`, `owner`, and `id`. Consequently, the `Address` interface includes a `get` and `set` method signature for each of the elements. For example, the interface includes `getTitle()` for retrieving data in the `<title>` element and `setTitle()` for modifying data in this element.

You can compile the Java source files with `javac`:

```
cd $ORACLE_HOME/xdk/demo/java/jaxb/Sample10/jaxbcustomized/sample10
javac *.java
```

Example 18-6 BusinessType.java

```
package jaxbcustomized.sample10;

public interface BusinessType
{
   public void setTitle(java.lang.String t);
   public java.lang.String getTitle();
   public void setOwner(java.lang.String o);
   public java.lang.String getOwner();
   public void setId(java.math.BigInteger i);
   public java.math.BigInteger getId();
}
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

18.3.2.3 Processing the XML Data in sample10.xml

`Sample10.java` unmarshals an XML document, prints its content, and marshals the XML to standard output.

`Sample10.java` shows how you can process the data in the `sample10.xml` document by using the class files that you generated in Generating and Compiling the Java Classes.

The `Sample10.java` program processes the XML data as follows:

1. Create strings for the XML data document file name and the name of the directory that contains the generated classes. This name is the package name. For example:

   ```
   String fileName = "sample10.xml";
   String instancePath = "jaxbcustomized.sample10";
   ```

2. Instantiate a JAXB context by invoking the `JAXBContext.newInstance()` method. This statement shows this technique:

   ```
   JAXBContext jc = JAXBContext.newInstance(instancePath);
   ```

3. Create the unmarshaller. This statement shows this technique:

   ```
   Unmarshaller u = jc.createUnmarshaller();
   ```

4. Unmarshal the XML document. The program unmarshals the document twice: it first returns an `Object` and then uses a cast to return a `MyBusiness` object. This statement shows this technique:

   ```
   Object obj = u.unmarshal(fileToURL(fileName));
   jaxbcustomized.sample10.MyBusiness bus =
            (jaxbcustomized.sample10.MyBusiness) u.unmarshal(fileToURL(fileName));
   ```

5. Print the contents of the XML document. The program invokes the `get` methods on the `MyBusiness` object. This code fragment shows this technique:

   ```
   System.out.println("My business details are: ");
   System.out.println("    title: " + bus.getTitle());
   System.out.println("    owner: " + bus.getOwner());
   System.out.println("    id:    " + bus.getId().toString());
   System.out.println();
   ```

6. Create a marshaller. This statement shows this technique:

   ```
   Marshaller m = jc.createMarshaller();
   ```

7. Configure the marshaller. You can invoke `setProperty()` to configure various properties the marshaller. The `JAXB_FORMATTED_OUTPUT` constant specifies that the marshaller must format the resulting XML data with line breaks and indentation. This statements show this technique:

   ```
   m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, new Boolean(true));
   ```

8. Marshal the content tree. This statement serializes the XML data, including markup, as an output stream:

   ```
   m.marshal(bus, System.out);
   ```

   By default, the marshaller uses UTF-8 encoding when writing XML data to an output stream.

Copyright © 1999, 2025, Oracle and/or its affiliates.

19

Using the XML Pipeline Processor for Java

An explanation is given of how to use the Extensible Markup Language (XML) pipeline processor for Java.

19.1 Introduction to the XML Pipeline Processor

Topics here include prerequisites, standards and specifications, multistage processing, and customized pipeline processing.

19.1.1 Prerequisites for Using the XML Pipeline Processor for Java

Prerequisites for using the XML Pipeline processor are listed.

This chapter assumes that you are familiar with these topics:

- XML Pipeline Definition Language. This XML vocabulary enables you to describe the processing relations between XML resources. For a more thorough introduction to the Pipeline Definition Language, consult the XML resources listed in Related Documents.
- Document Object Model (DOM). DOM is an in-memory tree representation of the structure of an XML document.
- Simple API for XML (SAX). SAX is a standard for event-based XML parsing.
- XML Schema language. See Using the XML Schema Processor for Java for an overview and links to suggested reading.

19.1.2 Standards and Specifications for the XML Pipeline Processor for Java

The Oracle XML Pipeline processor is based on the World Wide Web Consortium (W3C) XML Pipeline Definition Language Version 1.0 Note. The W3C Note defines an XML vocabulary rather than an application programming interface (API).

Pipeline Definition Language Standard for XDK for Java describes the differences between the W3C Note and the Oracle XML Developer's Kit (XDK) implementation of the Oracle XML Pipeline processor.

See Also

- *XML Pipeline Definition Language Version 1.0*
- Table 34-1

Copyright © 1999, 2025, Oracle and/or its affiliates.

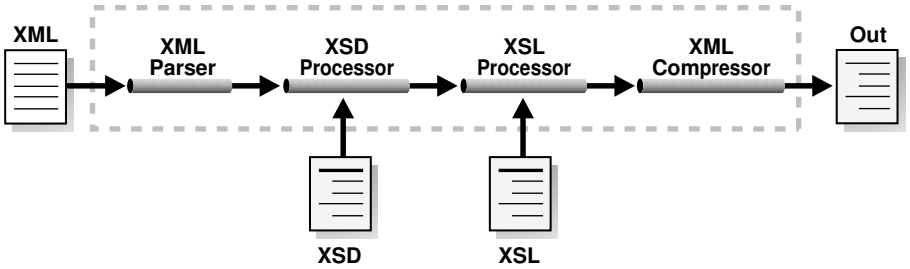
19.1.3 Multistage XML Processing

The Oracle XML Pipeline processor is built on the XML Pipeline Definition Language. The processor can take an input XML pipeline document and execute pipeline processes according to derived dependencies.

A **pipeline document**, which is written in XML, specifies the processes to be executed in a declarative manner. You can associate Java classes with processes by using the `<processdef/>` element in the pipeline document.

Use the Pipeline processor for multistage processing, which occurs when you process XML components sequentially or in parallel. The output of one stage of processing can become the input of another stage of processing. You can write a pipeline document that defines the inputs and outputs of the processes. Figure 19-1 shows a possible pipeline sequence.

Figure 19-1 Pipeline Processing



In addition to the XML Pipeline processor itself, XDK provides an API for processes that you can pipe together in a pipeline document. Table 19-2 summarizes the classes provided in the `oracle.xml.pipeline.processes` package.

The typical stages of processing XML in a pipeline are:

1. Parse the input XML documents. The `oracle.xml.pipeline.processes` package includes `DOMParserProcess` for DOM parsing and `SAXParserProcess` for SAX parsing.
2. Validate the input XML documents.
3. Serialize or transform the input documents. The Pipeline processor does not enable you to connect the SAX parser to the Extensible Stylesheet Language Transformation (XSLT) processor, which requires a DOM.

In multistage processing, SAX is ideal for filtering and searching large XML documents. Use DOM to change or access XML content efficiently and dynamically.

> **See Also**
>
> Processing XML in a Pipeline to learn how to write a pipeline document that provides the input for a pipeline application

Copyright © 1999, 2025, Oracle and/or its affiliates.

19.1.4 Customized Pipeline Processes

Class `oracle.xml.pipeline.controller.Process` is the base class for all pipeline process definitions. The classes in package `oracle.xml.pipeline.processes` extend this base class. To create a customized pipeline process, you must create a class that extends class `Process`.

At the minimum, every custom process must override the do-nothing `initialize()` and `execute()` methods of the `Process` class. If the customized process accepts SAX events as input, then it should override the `SAXContentHandler()` method to return the appropriate `ContentHandler` that handles incoming SAX events. It should also override the `SAXErrorHandler()` method to return the appropriate `ErrorHandler`. Table 19-1 provides further descriptions of the preceding methods.

Table 19-1 Methods in Class oracle.xml.pipeline.controller.Process

Class	Description
`initialize()`	Initializes the process before execution. Invoke `getInput()` to fetch a specific input object associated with the process element and invoke `supportType()` to indicate the types of input supported. Analogously, invoke `getOutput()` and `supportType()` for output.
`execute()`	Executes the process. Invoke `getInParaValue()`, `getInput()`, or `getInputSource()` to fetch the inputs to the process. If a custom process outputs SAX events, then it should invoke the `getSAXContentHandler()` and `getSAXErrorHandler()` methods in `execute()` to get the SAX handlers of these processes in the pipeline: Invoke `setOutputResult()`, `getOutputStream()`, `getOutputWriter()` or `setOutParam()` to set the outputs or outparams generated by this process. Invoke `getErrorSource()`, `getErrorStream()`, or `getErrorDocument()` to access the pipeline error element associated with this process element. If an exception occurs during `execute()`, invoke `error()` or `info()` to propagate it to the `PipelineErrorHandler`.
`SAXContentHandler()`	Returns the SAX `ContentHandler`. If dependencies from other processes are not available, then return `null`. When these dependencies are available, the method is executed till the end.
`SAXErrorHandler()`	Returns the SAX `ErrorHandler`. If you do not override this method, then the JAXB processor uses the default error handler implemented by this class to handle SAX errors.

> **See Also**
>
> *Oracle AI Database XML Java API Reference* for information about package `oracle.xml.pipeline.processes`

19.2 Using the XML Pipeline Processor for Java: Overview

Topics here include the basic process, running the demo programs, and using the XML pipeline processor command-line utility.

Copyright © 1999, 2025, Oracle and/or its affiliates.

19.2.1 Using the XML Pipeline Processor for Java: Basic Process

The basic process of the XML Pipeline Processor for Java is described.

The XML Pipeline processor for Java is accessible through these packages:

- `oracle.xml.pipeline.controller`, which provides an XML Pipeline controller that executes XML processes in a pipeline based on dependencies.
- `oracle.xml.pipeline.processes`, which provides wrapper classes for XML processes that can be executed by the XML Pipeline controller. The `oracle.xml.pipeline.processes` package contains the classes that you can use to design a pipeline application framework. Each class extends the `oracle.xml.pipeline.controller.Process` class.

  Table 19-2 lists the components in the package. You can connect these components and processes through a combination of the XML Pipeline processor and a pipeline document.

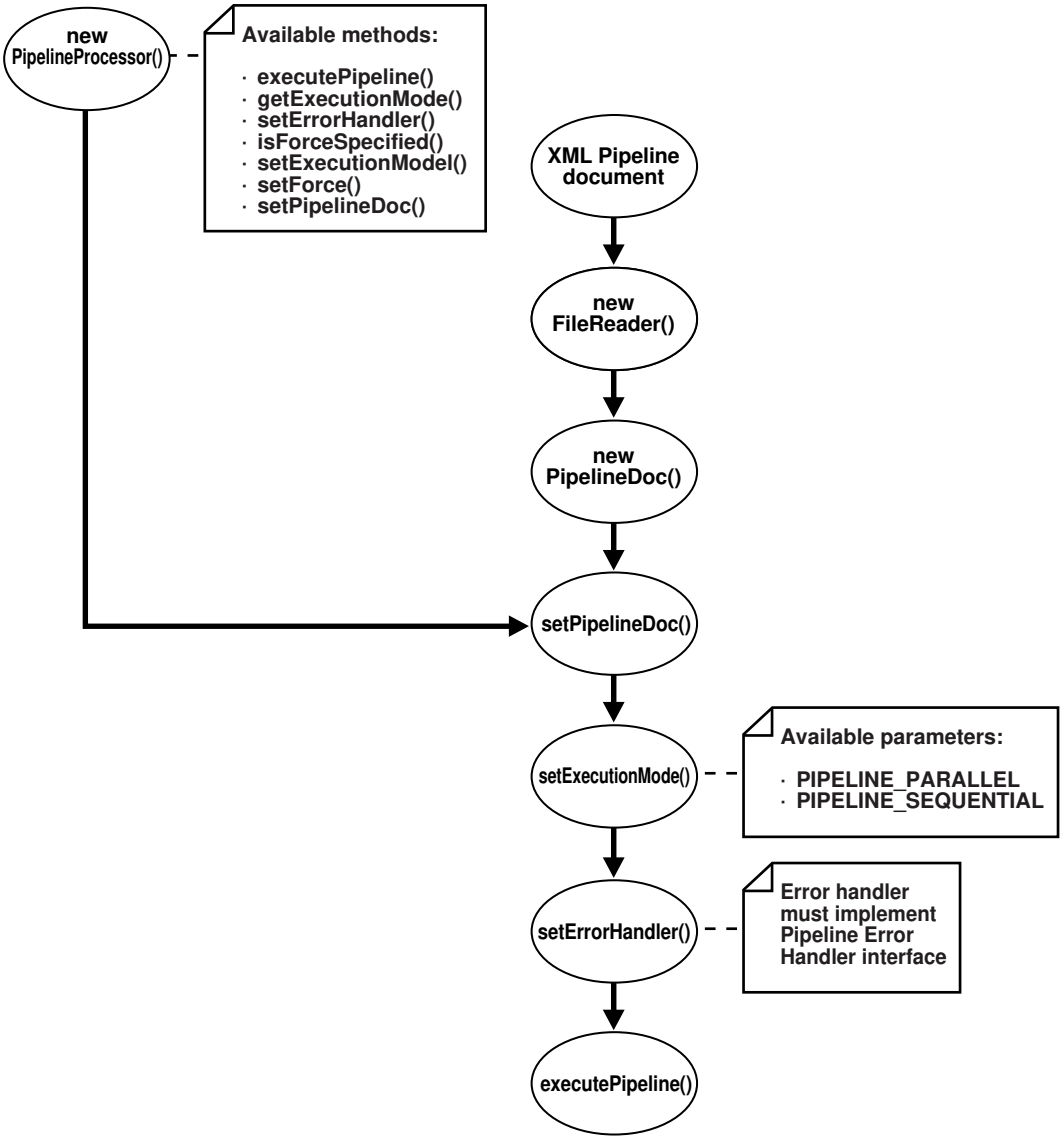
Table 19-2 Classes in oracle.xml.pipeline.processes

Class	Description
`CompressReaderProcess`	Receives compressed XML and outputs parsed XML.
`CompressWriterProcess`	Receives XML parsed with DOM or SAX and outputs compressed XML.
`DOMParserProcess`	Parses incoming XML and outputs a DOM tree.
`SAXParserProcess`	Parses incoming XML and outputs SAX events.
`XPathProcess`	Accepts a DOM as input, uses an XPath pattern to select one or more nodes from an XML `Document` or an XML `DocumentFragment`, and outputs a `Document` or `DocumentFragment`.
`XSDSchemaBuilder`	Parses an XML schema and outputs a schema object for validation. This process is built into the XML Pipeline processor and builds schema objects used for validating XML documents.
`XSDValProcess`	Validates against a local schema, analyzes the results, and reports errors if necessary.
`XSLProcess`	Accepts DOM as input, applies an XSL stylesheet, and outputs the result of the transformation.
`XSLStylesheetProcess`	Receives an XSL stylesheet as a stream or DOM and creates an `XSLStylesheet` object.

Figure 19-2 shows how to pass a pipeline document to a Java application that uses the XML Pipeline processor, configure the processor, and execute the pipeline.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Figure 19-2 Using the Pipeline Processor for Java



The basic steps are:

1. Instantiate a pipeline document, which forms the input to the pipeline execution. Create the object by passing a `FileReader` to the constructor:

   ```
   PipelineDoc pipe;
   FileReader f;
   pipe = new PipelineDoc((Reader)f, false);
   ```

2. Instantiate a pipeline processor. `PipelineProcessor` is the top-level class that executes the pipeline. Table 19-3 describes some available methods.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 19-3 PipelineProcessor Methods

Method	Description
`executePipeline()`	Executes the pipeline based on the `PipelineDoc` set by invoking `setPipelineDoc()`.
`getExecutionMode()`	Gets the type of execution mode: `PIPELINE_SEQUENTIAL` or `PIPELINE_PARALLEL`.
`setErrorHandler()`	Sets the error handler for the pipeline. This invocation is mandatory to execute the pipeline.
`setExecutionMode()`	Sets the execution mode. `PIPELINE_PARALLEL` is the default and specifies that the processes in the pipeline must execute in parallel. `PIPELINE_SEQUENTIAL` specifies that the processes in the pipeline must execute sequentially.
`setForce()`	Sets execution behavior. If `TRUE`, then the pipeline executes regardless of whether the target is up-to-date with the pipeline inputs.
`setPipelineDoc()`	Sets the `PipelineDoc` object for the pipeline.

This statement instantiates the pipeline processor:

```
proc = new PipelineProcessor();
```

3. Set the processor to the pipeline document. For example:

   ```
   proc.setPipelineDoc(pipe);
   ```

4. Set the execution mode for the processor and perform any other needed configuration. For example, set the mode by passing a constant to `PipelineProcessor.setExecutionMode()`.

   This statement specifies sequential execution:

   ```
   proc.setExecutionMode(PipelineConstants.PIPELINE_SEQUENTIAL);
   ```

5. Instantiate an error handler. The error handler must implement the `PipelineErrorHandler` interface. For example:

   ```
   errHandler = new PipelineSampleErrHdlr(logname);
   ```

6. Set the error handler for the processor by invoking `setErrorHandler()`. For example:

   ```
   proc.setErrorHandler(errHandler);
   ```

7. Execute the pipeline. For example:

   ```
   proc.executePipeline();
   ```

> **See Also**
>
> *Oracle AI Database XML Java API Reference* to learn about the `oracle.xml.pipeline` subpackages

Copyright © 1999, 2025, Oracle and/or its affiliates.

Related Topics

- Creating a Pipeline Document
  To use the Oracle XML Pipeline processor, you must create an XML document according to the rules of the Pipeline Definition Language specified in the W3C Note. The W3C specification defines the XML processing components and the inputs and outputs for these processes.

19.2.2 Running the XML Pipeline Processor Demo Programs

Demo programs for the XML Pipeline processor are included in `$ORACLE_HOME/xdk/demo/java/pipeline`.

Table 19-4 describes the XML files and Java source files that you can use to test the utility.

Table 19-4 Pipeline Processor Sample Files

File	Description
`README`	A text file that describes how to set up the Pipeline processor demos.
`PipelineSample.java`	A sample Pipeline processor application. The program takes `pipedoc.xml` as its first argument.
`PipelineSampleErrHdlr.java`	A sample program to create an error handler used by `PipelineSample`.
`book.xml`	A sample XML document that describes a series of books. This document is specified as an input by `pipedoc.xml`, `pipedoc2.xml`, and `pipedocerr.xml`.
`book.xsl`	An XSLT stylesheet that transforms the list of books in `book.xml` into an HTML table.
`book_err.xsl`	An XSLT stylesheet specified as an input by the `pipedocerr.xml` pipeline document. This stylesheet contains an intentional error.
`id.xsl`	An XSLT stylesheet specified as an input by the `pipedoc3.xml` pipeline document.
`items.xsd`	An XML schema document specified as an input by the `pipedoc3.xml` pipeline document.
`pipedoc.xml`	A pipeline document. This document specifies that process p1 must parse `book.xml` with DOM, process p2 must parse `book.xsl` and create a stylesheet object, and process p3 must apply the stylesheet to the DOM to generate `myresult.html`.
`pipedoc2.xml`	A pipeline document. This document specifies that process p1 must parse `book.xml` with SAX, process p2 must generate compressed XML `compxml` from the SAX events, and process p3 must regenerate the XML from the compressed stream as `myresult2.html`.
`pipedoc3.xml`	A pipeline document. This document specifies that a process p5 must parse `po.xml` with DOM, process p1 must select a single node from the DOM tree with an XPath expression, process p4 must parse `items.xsd` and generate a schema object, process p6 must validate the selected node against the schema, process p3 must parse `id.xsl` and generate a stylesheet object, and validated node to produce `myresult3.html`.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 19-4 (Cont.) Pipeline Processor Sample Files

File	Description
`pipedocerr.xml`	A pipeline document. This document specifies that process p1 must parse `book.xml` with DOM, process p2 must parse `book_err.xsl` and generate a stylesheet object if it encounters no errors and apply an inline stylesheet if it encounters errors, and process p3 must apply the stylesheet to the DOM to generate `myresulterr.html`. Because `book_err.xsl` contains an error, the program must write the text contents of the input XML to `myresulterr.html`.
`po.xml`	A sample XML document that describes a purchase order. This document is specified as an input by `pipedoc3.xml`.

Documentation for how to compile and run the sample programs is located in the `README`. The basic steps are:

1. Change into the `$ORACLE_HOME/xdk/demo/java/pipeline` directory (UNIX) or `%ORACLE_HOME%\xdk\demo\java\pipeline` directory (Windows).
2. Ensure that your environment variables are set as described in Setting Up the XDK for Java Environment.
3. Run `make` (UNIX) or `Make.bat` (Windows) at the system prompt to generate class files for `PipelineSample.java` and `PipelineSampleErrHdler.java` and run the demo programs. The programs write output files to the `log` subdirectory.

   Alternatively, you can run the demo programs manually by using this syntax:

   ```
   java PipelineSample pipedoc pipelog [ seq | para ]
   ```

   The *`pipedoc`* option specifies which pipeline document to use. The *`pipelog`* option specifies the name of the pipeline log file, which is optional unless you specify `seq` or `para`, in which case a file name is required. If you do not specify a log file, then the program generates `pipeline.log` by default. The `seq` option processes threads sequentially; `para` processes in parallel. If you specify neither `seq` or `para`, then the default is parallel processing.
4. View the files generated from the pipeline, which are all named with the initial string `myresult`, and the log files.

19.2.3 Using the XML Pipeline Processor Command-Line Utility

The command-line interface for the XML Pipeline processor is named `orapipe`. The Pipeline processor is packaged with Oracle Database. By default, the Oracle Universal Installer installs the utility on disk in `$ORACLE_HOME/bin`.

Before running the utility for the first time, ensure that your environment variables are set as described in Setting Up the XDK for Java Environment. Run `orapipe` at the operating system command line with this syntax:

```
orapipe options pipedoc
```

The *`pipedoc`* is the pipeline document, which is required. Table 19-5 describes the available options for the `orapipe` utility.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 19-5 orapipe Command-Line Options

Option	Purpose
`-help`	Prints the help message
`-log` *`logfile`*	Writes errors and messages to the specified log file. The default is `pipeline.log`.
`-noinfo`	Does not log informational items. The default is on.
`-nowarning`	Does not log warnings. The default is on.
`-validate`	Validates the input `pipedoc` with the pipeline schema. Validation is turned off by default. If `outparam` feature is used, then `validate` fails with the current pipeline schema because this is an additional feature.
`-version`	Prints the release version.
`-sequential`	Executes the pipeline in sequential mode. The default is parallel.
`-force`	Executes pipeline even if target is up-to-date. By default no force is specified.
`-attr` *`name value`*	Sets the value of $*`name`* to the specified *`value`*. For example, if the attribute name is `source` and the value is `book.xml`, then you can pass this value to an element in the pipeline document: `<input ... label="$source">`.

19.3 Processing XML in a Pipeline

Topics here include creating a pipeline document, writing a pipeline processor application, and writing a pipeline error handler.

19.3.1 Creating a Pipeline Document

To use the Oracle XML Pipeline processor, you must create an XML document according to the rules of the Pipeline Definition Language specified in the W3C Note. The W3C specification defines the XML processing components and the inputs and outputs for these processes.

The XML Pipeline processor includes support for these XDK components:

- XML parser
- XML compressor
- XML Schema validator
- XSLT processor

19.3.1.1 Example of a Pipeline Document

The XML Pipeline processor executes a sequence of XML processing according to the rules in the pipeline document and returns a result. The sample pipeline document that is included in the demo directory is presented.

Example 19-1 pipedoc.xml

```
<pipeline xmlns="http://www.w3.org/2002/02/xml-pipeline"
          xml:base="http://example.org/">

  <param name="target" select="myresult.html"/>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
<processdef name="domparser.p"
 definition="oracle.xml.pipeline.processes.DOMParserProcess"/>
<processdef name="xslstylesheet.p"
 definition="oracle.xml.pipeline.processes.XSLStylesheetProcess"/>
<processdef name="xslprocess.p"
 definition="oracle.xml.pipeline.processes.XSLProcess"/>

 <process id="p2" type="xslstylesheet.p" ignore-errors="false">
   <input name="xsl" label="book.xsl"/>
   <outparam name="stylesheet" label="xslstyle"/>
 </process>

 <process id="p3" type="xslprocess.p" ignore-errors="false">
   <param name="stylesheet" label="xslstyle"/>
   <input name="document" label="xmldoc"/>
   <output name="result" label="myresult.html"/>
 </process>

<process id="p1" type="domparser.p" ignore-errors="true">
   <input name="xmlsource" label="book.xml "/>
   <output name="dom" label="xmldoc"/>
   <param name="preserveWhitespace" select="true"></param>
   <error name="dom">
     <html xmlns="http://www/w3/org/1999/xhtml">
       <head>
          <title>DOMParser Failure!</title>
       </head>
       <body>
         <h1>Error parsing document</h1>
       </body>
     </html>
   </error>
</process>

</pipeline>
```

19.3.1.1.1 Processes Specified in the Pipeline Document

The processes specified in the pipeline document are described.

In Example 19-1, three processes are called and associated with Java classes in the `oracle.xml.pipeline.processes` package. The pipeline document uses element `processdef` to make these associations:

- `domparser.p` is associated with the `DOMParserProcess` class
- `xslstylesheet.p` is associated with the `XSLStylesheetProcess` class
- `xslprocess.p` is associated with the `XSLProcess` class

19.3.1.1.2 Processing Architecture Specified in the Pipeline Document

The basic design of and the processing architecture of the pipeline are described.

The `PipelineSample` program accepts the `pipedoc.xml` document shown in Example 19-1 as input along with XML documents `book.xml` and `book.xsl`. The basic design of the pipeline is:

1. Parse the incoming `book.xml` document and generate a DOM tree. This task is performed by `DOMParserProcess`.
2. Parse `book.xsl` as a stream and generate an `XSLStylesheet` object. This task is performed by `XSLStylesheetProcess`.

Copyright © 1999, 2025, Oracle and/or its affiliates.

3. Receive the DOM of `book.xml` as input, apply the stylesheet object, and write the result to `myresult.html`. This task is performed by `XSLProcess`.

Note these aspects of the processing architecture used in the pipeline document:

- The target information set, `http://example.org/myresult.html`, is inferred from the default value of the `target` parameter and the `xml:base` setting.
- The process `p2` has an input of `book.xsl` and an output parameter with the label `xslstyle`, so it must run to produce the input for `p3`.
- The `p3` process depends on input parameter `xslstyle` and document `xmldoc`.
- The `p3` process has an output parameter with the label `http://example.org/myresult.html`, so it must run to produce the target.
- The process `p1` depends on input document `book.xml` and outputs `xmldoc`, so it must run to produce the input for `p3`.

In Example 19-1, more than one order of processing can satisfy all of the dependencies. Given the rules, the XML Pipeline processor must process `p3` last but can process `p1` and `p2` in either order or process them in parallel.

19.3.2 Writing a Pipeline Processor Application

The `PipelineSample.java` source file shows a basic pipeline application.

You can use the application with any of the pipeline documents in Table 19-4 to parse and transform an input XML document.

The basic steps of the program are:

1. Perform the initial setup. The program declares references of type `FileReader` (for the input XML file), `PipelineDoc` (for the input pipeline document), and `PipelineProcessor` (for the processor). The first argument is the pipeline document, which is required. If a second argument is received, then it is stored in the `logname` String. This code fragment shows this technique:

```
public static void main(String[] args)
{
  FileReader f;
  PipelineDoc pipe;
  PipelineProcessor proc;

  if (args.length < 1)
  {
    System.out.println("First argument needed, other arguments are ".
                       "optional:");
    System.out.println("pipedoc.xml <output_log> <'seq'>");
    return;
  }
  if (args.length > 1)
    logname = args[1];
  ...
```

2. Create a `FileReader` object by passing the first command-line argument to the constructor as the file name. For example:

```
f = new FileReader(args[0]);
```

3. Create a `PipelineDoc` object by passing the reference to the `FileReader` object. This example casts the `FileReader` to a `Reader` and specifies no validation:

Copyright © 1999, 2025, Oracle and/or its affiliates.

```
pipe = new PipelineDoc((Reader)f, false);
```

4. Instantiate an XML Pipeline processor. This statement instantiates the pipeline processor:

```
proc = new PipelineProcessor();
```

5. Set the processor to the pipeline document. For example:

```
proc.setPipelineDoc(pipe);
```

6. Set the execution mode for the processor and perform any other configuration. This code fragment uses a condition to determine the execution mode. If three or more arguments are passed to the program, then it sets the mode to sequential or parallel depending on which argument is passed. For example:

```
String execMode = null;
if (args.length > 2)
{
   execMode = args[2];
   if(execMode.startsWith("seq"))
      proc.setExecutionMode(PipelineConstants.PIPELINE_SEQUENTIAL);
   else if (execMode.startsWith("para"))
      proc.setExecutionMode(PipelineConstants.PIPELINE_PARALLEL);
}
```

7. Instantiate an error handler. The error handler must implement the `PipelineErrorHandler` interface. The program uses the `PipelineSampleErrHdler` shown in `PipelineSampleErrHdlr.java`. This code fragment shows this technique:

```
errHandler = new PipelineSampleErrHdlr(logname);
```

8. Set the error handler for the processor by invoking `setErrorHandler()`. This statement shows this technique:

```
proc.setErrorHandler(errHandler);
```

9. Execute the pipeline. This statement shows this technique:

```
proc.executePipeline();
```

> **See Also**
>
> *Oracle AI Database XML Java API Reference* to learn about the `oracle.xml.pipeline` subpackages

19.3.3 Writing a Pipeline Error Handler

An application invoking the XML Pipeline processor must implement the `PipelineErrorHandler` interface to handle errors received from the processor. Set the error handler in the processor by invoking `setErrorHandler()`. When writing the error handler, you can choose to throw an exception for different types of errors.

The `oracle.xml.pipeline.controller.PipelineErrorHandler` interface declares the methods shown in Table 19-6, all of which return `void`.

Table 19-6 PipelineErrorHandler Methods

Method	Description
`error(java.lang.String msg, PipelineException e)`	Handles `PipelineException` errors.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 19-6 (Cont.) PipelineErrorHandler Methods

Method	Description
`fatalError(java.lang.String msg, PipelineException e)`	Handles fatal `PipelineException` errors.
`warning(java.lang.String msg, PipelineException e)`	Handles `PipelineException` warnings.
`info(java.lang.String msg)`	Prints optional, additional information about errors.

The first three methods in Table 19-6 receive a reference to an `oracle.xml.pipeline.controller.PipelineException` object. These methods of the `PipelineException` class are especially useful:

- `getExceptionType()`, which gets the type of exception thrown
- `getProcessId()`, which gets the process ID where the exception occurred
- `getMessage()`, which returns the message string of this `Throwable` error

The `PipelineSampleErrHdler.java` source file implements a basic error handler for use with the `PipelineSample` program. The basic steps are:

1. Implement a constructor. The constructor accepts the name of a log file and wraps it in a `FileWriter` object:

```
PipelineSampleErrHdlr(String logFile) throws IOException
{
  log = new PrintWriter(new FileWriter(logFile));
}
```

2. Implement the `error()` method. This implementation prints the process ID, exception type, and error message. It also increments a variable holding the error count. For example:

```
public void error (String msg, PipelineException e) throws Exception
{
  log.println("\nError in: " + e.getProcessId());
  log.println("Type: " + e.getExceptionType());
  log.println("Message: " +  e.getMessage());
  log.println("Error message: " + msg);
  log.flush();
  errCount++;
}
```

3. Implement the `fatalError()` method. This implementation follows the pattern of `error()`. For example:

```
public void fatalError (String msg, PipelineException e) throws Exception
{
  log.println("\nFatalError in: " + e.getProcessId());
  log.println("Type: " + e.getExceptionType());
  log.println("Message: " +  e.getMessage());
  log.println("Error message: " + msg);
  log.flush();
  errCount++;
}
```

4. Implement the `warning()` method. This implementation follows the basic pattern of `error()` except it increments the `warnCount` variable rather than the `errCount` variable. For example:

```
public void warning (String msg, PipelineException e) throws Exception
{
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
  log.println("\nWarning in: " + e.getProcessId());
  log.println("Message: " +  e.getMessage());
  log.println("Error message: " + msg);
  log.flush();
  warnCount++;
}
```

5. Implement the `info()` method. Unlike the preceding methods, this method does not receive a `PipelineException` reference as input. This implementation prints the `String` received by the method and increments the value of the `warnCount` variable:

```
public void info (String msg)
{
  log.println("\nInfo : " + msg);
  log.flush();
  warnCount++;
}
```

6. Implement a method to close the `PrintWriter`. This code implements the method `closeLog()`, which prints the number of errors and warnings and invokes `PrintWriter.close()`:

```
public void closeLog()
{
  log.println("\nTotal Errors: " + errCount + "\nTotal Warnings: " +
              warnCount);
  log.flush();
  log.close();
}
```

> **See Also**
>
> *Oracle AI Database XML Java API Reference* to learn about the `PipelineErrorHandler` interface and the `PipelineException` class

Copyright © 1999, 2025, Oracle and/or its affiliates.

20

Determining XML Differences Using Java

An explanation is given of how to determine the differences between two Extensible Markup Language (XML) inputs, using the Java library included in the Oracle XML Developer's Kit (XDK).

20.1 Overview of XML Diffing Utilities for Java

The Java XML diffing library includes diffing, hashing, and equality-comparison methods for XML inputs in class `XmlUtils` of package `oracle.xml.diff`.

The `Options` class in the `oracle.xml.diff` package provides options that enable users to control how the input is processed by the methods in the `XmlUtils` class (see User Options for the Java XML Diffing Library). One of these supported options is white space normalization, which is enabled by default.

The algorithm used by the XML diffing methods is specifically designed for the use case of finding differences between two large XML documents (5 MB or more) within seconds, where the minimal diff is not required. The minimal diff is the smallest possible set of changes which, when applied to the first XML input, produces an output equivalent (identical) to the second XML input. Known minimal diff algorithms require prohibitively large amounts of memory and time for processing multimegabyte inputs. The algorithm used in the XML diff methods produces best quality (as close to minimal as possible) diffs in the absence of recurring identical subtrees in the XML inputs.

The Java XML diffing library provides several equivalent variants of each method to allow XML inputs in different forms, including Document Object Model (DOM) nodes, files, and input streams. Internally, the diffing, hashing, and equality comparisons operate on a DOM tree. Input that is not in the form of a DOM tree is internally converted to a DOM tree. To reduce computational overhead, Oracle recommends passing in DOM directly whenever possible.

The Java XML diffing library includes methods to return the diff output as a DOM document, or as a list of objects, each representing a diff operation. With the second option, you can avoid the overhead of XML document generation. With the first option, the resulting document conforms to the XML schema described in Diff Output Schema. The first option is useful, for example, if the diff output must be stored as a log for future reference.

The hash methods provided by the Java XML diffing library compute the hash value of XML input. If the hash values of the two XML inputs are equal, they are identical with a very high probability.

The equal methods provided in the Java XML diffing library compare two inputs for equality.

To use the Java XML diffing library, your application must run with Java version 1.6 or later, with any DOM implementation.

Copyright © 1999, 2025, Oracle and/or its affiliates.

> **Note**
>
> The application programming interface (API) components described in this chapter are contained within the Java package `oracle.xml.diff`. For brevity, fully qualified names are used only when necessary to avoid confusion.
>
> See *Oracle AI Database XML Java API Reference* for more information about the `oracle.xml.diff` package.

20.2 User Options for the Java XML Diffing Library

The Java XML diffing library supports two options, which you can set using methods in the `Options` class of the `oracle.xml.diff` package. The `Options` object is passed in directly to the diff, hash, and equal methods on each invocation.

- Text Node Normalization (enabled by default)

  Text nodes are normalized in the DOM trees on which the diff, hash, and equal methods operate. Text node normalization involves coalescing adjacent text nodes, followed by stripping leading and trailing white space from the coalesced nodes. Single text nodes have their leading and trailing white space stripped. White-space-only text nodes are eliminated.

  Normalization is performed within the library with minimal additional space, and without modifying the provided XML inputs.

  To perform your own normalization on the DOM inputs before passing them to the library, you must invoke the method `normalizeTextNodes(false)` on the `Options` object to turn off the default normalization.

  Oracle does not recommend invoking the diff methods without performing some type of normalization, either the default or your own. The diff quality suffers in the presence of identical white space text nodes, which commonly occur in XML documents.

- Ignoring Namespace Prefix Differences (enabled by default)

  XML namespace prefix differences are ignored by the diff, hash, and equal methods. For example, two DOM nodes are considered equal if they are identical except for having different prefixes (even if the two different prefixes map to Universal Resource Identifier (URI) of the same namespace. To configure the library to treat different namespace prefixes as truly different, even if they map to the same URI, you can invoke the method `ignorePrefixDifferences(false)` on the `Options` object to turn off the default namespace prefix behavior.

> **See Also**
>
> *Oracle AI Database XML Java API Reference* for details about the methods in the `Options` class

Copyright © 1999, 2025, Oracle and/or its affiliates.

20.3 Using Java XML Diffing Methods to Find Differences

The Java XML dffing library provides various `diff` and `diffToDoc` methods in the `XmlUtils` class of the `oracle.xml.diff` package. You can use these methods to compare two XML inputs to determine if there are any differences between them.

The `diffToDoc` methods return the output as a DOM document that conforms to the schema described in Diff Output Schema. The Java XML diffing library includes several equivalent variants of these methods, which accept inputs in different forms (DOM nodes, files, and others).

The Java XML diffing library includes an equivalent set of `diff` methods that enable you to work on the diff output that is returned as a list of diff operation objects.

Because the DOM document that represents the diff does not need to be constructed, using the `diff` methods is more efficient than using the `difftoDoc` methods. You should consider using these methods whenever you do not need a representation of the diff in XML form. To use the `diff` methods, you must create an implementation of the `DiffOpReceiver` interface, and then pass it as a parameter to the `diff` methods. The `DiffOpReceiver.receiveDiff` method receives the diff as a list of `DiffOp` objects.

The diff result, whether it is returned as a DOM document or as a list of `DiffOps` objects, can be understood as a series of diff operations. The possible diff operations are:

- append-node
- insert-node-before
- delete-node

Applying the sequence of diff operations on the first DOM tree produces a tree that is equivalent to the second DOM tree. For example, using these two XML inputs:

First input: `<a><b/></a>`

Second input: `<a><c/></a>`

The diff result from comparing the first and second input is a list, with these two diff operations:

```
delete-node /a[1]/b[1]
append-node <c/> to /a[1]
```

Deleting the node represented by the XPath expression `/a/b` in the first input, and then appending `<c/>` to the node represented by the XPath expression `/a` in the first input produces the result `<a><c/></a>`, which is equivalent to the second input.

When the diff operations are output to a DOM document by the `domToDoc(…)` method, they rely on XPath expressions to indicate the node locations. These XPath locations refer to node positions in the original first input. They do not reflect the applied diff operations.

Copyright © 1999, 2025, Oracle and/or its affiliates.

> **Note**
>
> The Java XML diffing library does not support append-node, insert-node-before, and delete-node operations for attribute nodes. Thus, when any attributes of a node are changed, the change is shown as a delete of the whole node, followed by the insert or the append of the new node with the changed attributes.
>
> For example, for these two inputs:
>
> First input: `<a attr1="val1"><b/></a>`
>
> Second input: `<a attr2="val2"><b/</a>`
>
> The diff consists of these two diff operations:
>
> ```
> insert <a attr2="val2"><b/></a> before /a[1]
> delete /a[1]
> ```

> **Note**
>
> This section uses XML document output to describe each diff operation. Although they are not described here, diff operation results that are returned programmatically are equivalent.

> **See Also**
>
> *Oracle AI Database XML Java API Reference* for more information about the `DiffOpReceiver` interface, and for details about the methods in the `XmlUtils` class

20.3.1 About the append-node Operation

The append-node operation specifies that a given node is to be appended as the last child of a particular first input node.

Example 20-1 shows an append-node operation that adds the highlighted node `<enumeration value="FL"/>` to a document.

Invoking a `diffToDoc(…)` method, using the original document (without the highlighted change) and the changed document as input produces this output:

```
<xd:append-node
 xd:parent-xpath="/schema[1]/simpleType[1]/restriction[1]"
 xd:node-type="element">
        <xd:content>
            <enumeration value="FL"/>
        </xd:content>
</xd:append-node>
```

The append-node operation is represented by the `<append-node>` element in the preceding output. This element specifies that a node of the given type is added as the last child of the

Copyright © 1999, 2025, Oracle and/or its affiliates.

given first input parent node. The `parent-xpath` attribute specifies the parent node. The `node-type` attribute specifies the type of the node to be appended. The `<content>` child element specifies the node to be appended.

Alternatively, when the `diff(…)` methods are used, the append-node operation is accessible in the `DiffOpReceiver.receiverDiff(…)` method as a `DiffOp` object. In this case, the operation returns the actual references to the nodes in the two DOM trees involved in the diff operation. The reference to the parent node in the first input is returned by invoking the `getParent()` method of `DiffOp`. The reference to the node to be appended from the second input is returned by invoking the `getNew()` method of `DiffOp`.

Example 20-1 Appending a Node

```
<schema>
…
     <simpleType name="USState">
          <restriction base="string">
               <enumeration value="NY"/>
               <enumeration value="TX"/>
               <enumeration value="CA"/>
               <enumeration value="FL"/>
          </restriction>
     </simpleType>
…
</schema>
```

20.3.2 About the insert-node-before Operation

The insert-node-before operation specifies that a given node is to be inserted before a particular node in the first input.

Example 20-2 shows an insert-node-before operation that inserts the highlighted node `<!-- A type representing US States -->` before the node `<simpleType name="USState">` in a document.

Invoking a `diffToDoc(…)` method, using the original document (without the highlighted change) and the changed document as input produces this output:

```
<xd:insert-node-before xd:node-type="comment"
 xd:xpath="/schema[1]/simpleType[1]">
           <xd:content>
                   <!-- A type representing US States -->
           </xd:content>
</xd:insert-node-before>
```

The insert-node-before operation is represented by the `<insert-node-before>` element in the preceding output. This element specifies that a node of the given type is inserted before the given first input node. The `xpath` attribute specifies the location of the first input node. The `node-type` attribute specifies the type of the node to be inserted. The `<content>` child element specifies the node to be inserted.

Alternatively, when the `diff(…)` methods are used, the insert-node-before operation is accessible in the `DiffOpReceiver.receiverDiff(…)` method as a `DiffOp` object. In this case, the operation returns the actual references to the nodes in the two DOM trees involved in the diff operation. The reference to the node before which to insert a node in the first input is returned by invoking the `getSibling()` method of `DiffOp`. The reference to the node to be inserted from the second input is returned by invoking the `getNew()` method of `DiffOp`.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Example 20-2 Inserting a Node

```
<schema>
…
    <!-- A type representing US States -->
    <simpleType name="USState">
            <restriction base="string">
                <enumeration value="NY"/>
                <enumeration value="TX"/>
                <enumeration value="CA"/>
            </restriction>
    </simpleType>
…
</schema>
```

20.3.3 About the delete-node Operation

The delete-node operation specifies that a particular node (and its subtree) in the first input is to be deleted.

Example 20-3 shows a delete-node operation that deletes the highlighted node `<element name="LineItems" maxOccurs="unbounded">` from a document.

Invoking a diffToDoc(…) method, using the original document (without the highlighted change) and the changed document as input produces this output:

```
<xd:delete-node xd:node-type="element" xd:xpath=
 "/schema[1]/element[1]/complexType[1]/sequence[1]/element[1]/element[1]"/>
```

The delete-node operation is represented by the `<delete-node>` element in the preceding output. This element specifies that a node of the given type is deleted. The `xpath` attribute specifies the location of the first input node. The `node-type` attribute specifies the type of the node to be deleted.

Alternatively, when the `diff(…)` methods are used, the delete-node operation is accessible in the `DiffOpReceiver.receiverDiff(…)` method as a `DiffOp` object. In this case, the operation returns the actual reference to the node in the first input DOM tree. The reference to the node to be deleted from the first input is returned by invoking `getCurrent()` method of `DiffOp`.

Example 20-3 Deleting a Node

```
<schema>
…
     <element name="PurchaseOrder">
        <complexType>
           <sequence>
              <element name="PO-Number" type="string">
                 <element name="LineItems" maxOccurs="unbounded">
…
</schema>
```

20.4 Invoking diff and difftoDoc Methods in a Java Application

Examples here show how to compare two inputs by invoking `diff` and `diffToDoc` methods from a Java application.

Example 20-4 shows how to use the `diffToDoc` method to compare the input files `doc` and `doc1`.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Continuing with this example, the two input files `f1.xml` and `f2.xml` contain the same data as in Example 20-1.

This sample code displays the contents of `f1.xml`:

```
<schema>
     <simpleType name="USState">
          <restriction base="string">
               <enumeration value="NY"/>
               <enumeration value="TX"/>
               <enumeration value="CA"/>
          </restriction>
     </simpleType>
</schema>
```

And this sample code displays the contents of `f2.xml`:

```
<schema>
     <simpleType name="USState">
          <restriction base="string">
               <enumeration value="NY"/>
               <enumeration value="TX"/>
               <enumeration value="CA"/>
               <enumeration value="FL"/>
          </restriction>
     </simpleType>
</schema>
```

Assume that `textDiff.java` and the input files are in the current directory. Then enter these commands to compile and run the example:

```
javac -classpath "xml.jar" textDiff.java
java –classpath “xml.jar:." textDiff f1.xml f2.xml
```

Serializing the resulting `diffAsDom` document produces this output:

```
<xd:xdiff xmlns:xd="http://xmlns.oracle.com/xdb/xdiff.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://
xmlns.oracle.com/xdb/xdiff.xsd http://xmlns.oracle.com/xdb/xdiff.xsd">
    <?oracle-xmldiff operations-in-docorder="true"
     output-model="snapshot" diff-algorithm="greedy-heuristic"?>
    <xd:append-node xd:node-type="element"
     xd:parent-xpath="/schema[1]/simpleType[1]/restriction[1]">
        <xd:content>
            <enumeration value="FL"/>
        </xd:content>
    </xd:append-node>
</xd:xdiff>
```

Example 20-5 shows how to use an implementation of the `DiffOpReceiver` interface to process the diff returned from the comparison between two XML inputs as a list of `DiffOp` objects.

Enter these commands to compile and run the example:

```
javac -classpath "xml.jar" progDiff.java
java –classpath “xml.jar:." progDiff f1.xml f2.xml
```

The example generates this output:

```
APPENDING NODE:
<enumeration value="FL"/>
TO THE PARENT NODE:
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
<restriction base="string">
    <enumeration value="NY"/>
    <enumeration value="TX"/>
    <enumeration value="CA"/>
</restriction>
```

Example 20-4 Getting a diff as a Document from a Java Application

```
import oracle.xml.diff.XmlUtils;
import oracle.xml.diff.Options;

import java.io.File;

import org.w3c.dom.Node;
import org.w3c.dom.Document;

import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;

public class textDiff
{
    public static void main(String[] args) throws Exception
    {
        XmlUtils xmlUtils = new XmlUtils();

        //Parse the two input files
        DocumentBuilderFactory dbFactory =
                  DocumentBuilderFactory.newInstance();
        dbFactory.setNamespaceAware(true);
        DocumentBuilder docBuilder =
                  dbFactory.newDocumentBuilder();
        Node doc = docBuilder.parse(new File(args[0]));
        Node doc1 = docBuilder.parse(new File(args[1]));

        //Run the diff
        try
        {
            Document diffAsDom = xmlUtils.diffToDoc(doc,
                                  doc1, new Options());
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

Example 20-5 Getting a diff Using DiffOpReceiver from a Java Application

```
import oracle.xml.diff.DiffOp;
import oracle.xml.diff.DiffOpReceiver;

import java.util.List;
import java.util.Properties;

import java.io.File;

import org.w3c.dom.Node;

import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;

public class progDiff
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
{
  public static void main(String[] args) throws Exception
  {
     XmlUtils xmlUtils = new XmlUtils();

     //Parse the two input files
     DocumentBuilderFactory dbFac =
                       DocumentBuilderFactory.newInstance();
     dbFac.setNamespaceAware(true);
     DocumentBuilder docBuilder = dbFac.newDocumentBuilder();
     Node doc = docBuilder.parse(new File(args[0]));
     Node doc1 = docBuilder.parse(new File(args[1]));

     Options opt = new Options();

     //Instantiate the DiffOpReceiver. This is the object that
     //will receive DiffOps, ie diff operations that the XmlDiff
     //outputs. Each object represents either deletion or insert
     //or append of a node. In this DiffOpReceiverImpl
     //implementation (see below) of the DiffOpReceiver
     //interface, we simply print out each diff operation.
     DiffOpReceiver diffOpRec =
                 new progDiff().new DiffOpReceiverImpl();
     xmlUtils.diff(doc, doc1, diffOpRec, opt);
  }

  class DiffOpReceiverImpl implements DiffOpReceiver
  {
      public void receiveDiff(List<DiffOp> diffOps)
      {
         try
         {
             for (int i = 0; i < diffOps.size(); i++)
             {
                 DiffOp diffOperation= diffOps.get(i);

                 //Delete operation, print out the deleted
                 // node from the first tree
                 if (diffOperation.getOpName() ==
                     DiffOp.Name.DELETE)
                    System.out.println ("DELETING NODE:\n" +
    XmlUtils.nodeToString(diffOperation.getCurrent(), false));

                 //Insert operation. Print out the node
                 //from the second tree to be inserted,
                 //and the node from the first tree
                 //before  which the insertion will happen
                 else if (diffOperation.getOpName() ==
                           DiffOp.Name.INSERT_BEFORE_NODE)
                    System.out.println ("INSERTING NODE:\n" +
       XmlUtils.nodeToString(diffOperation.getNew(), false) +
                                          "BEFORE NODE:\n" +
    XmlUtils.nodeToString(diffOperation.getSibling(), false));




                 //Append as the last node of the parent.
                 //Print out the node from the second tree
                 //that will be appended, and the parent
                 //node from the first tree to which the
                 //former node will be appended as the
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
                    //last child.
                    else if (diffOperation.getOpName() ==
                       DiffOp.Name.INSERT_BY_APPENDING)
                       System.out.println ("APPENDING NODE:\n" +
                XmlUtils.nodeToString(diffOperation.getNew(), false) +
                                          "TO THE PARENT NODE:\n" +
                XmlUtils.nodeToString(diffOperation.getParent(), false));
                    }
                }
                catch (Exception e)
                {
                    System.err.println ("Error while printing out the
                                    diff result:" + e.getMessage());
                }
              }
            }
          }
```

20.5 Using Java XML hash and equal Methods to Identify and Compare Inputs

The Java XML diffing library provides `hash` methods to compute a hash value that uniquely identifies the input, with a high probability. Because there is a very low probability of a hash collision, there can be no guarantee that two inputs are identical when their hash values match.

To check that two inputs are truly identical with absolute certainty, use the `equal` methods. The `equal` methods process both inputs simultaneously, while checking them for absolute equality.

The Java XML diffing library provides several equivalent variants of the `hash` and `equal` methods that accept inputs in different forms (DOM nodes, files, and more).

> **See Also**
>
> *Oracle AI Database XML Java API Reference* for details about the `hash` and `equal` methods in the `XmlUtils` class

20.6 Diff Output Schema

The output schema `xdiff.xsd`, to which the Java XML diffing library conforms, is presented.

Example 20-6 Diff Output Schema: xdiff.xsd

```
<schema targetNamespace="http://xmlns.oracle.com/xdb/xdiff.xsd"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:xd="http://xmlns.oracle.com/xdb/xdiff.xsd"
    version="1.0" elementFormDefault="qualified"
    attributeFormDefault="qualified">
    <annotation>
        <documentation xml:lang="en">
         Defines the structure of XML documents that capture the difference
         between two XML inputs. Changes that are not supported by Oracle
         XmlDiff may not be expressible in this schema.

        'oracle-xmldiff' PI:

        We use 'oracle-xmldiff' PI to describe certain aspects of the diff.
        This should be the first element of top level xdiff element.
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
        version-number: version number of the XML diff schema

        output-model: output model for representing the diff. Currently, only
        the "snapshot" model is supported.

        Snapshot model:
        Each operation uses XPaths as if no operations
        have been applied to the input document.
        Default and works for both Xmldiff and XmlPatch.

        <!-- Example:
            <?oracle-xmldiff version-number = "1.0" output-model = "snapshot"?>
        -->
        </documentation>
     </annotation>
    <!-- Enumerate the supported node types -->
    <simpleType name="xdiff-nodetype">
        <restriction base="string">
            <enumeration value="element"/>
            <enumeration value="text"/>
            <enumeration value="cdata"/>
            <enumeration value="processing-instruction"/>
            <enumeration value="comment"/>
         </restriction>
    </simpleType>

    <element name="xdiff">
        <complexType>
            <choice minOccurs="0" maxOccurs="unbounded">

                <element name="append-node">
                    <complexType>
                        <sequence>
                            <element name="content" type="anyType"/>
                        </sequence>
                        <attribute name="node-type" type="xd:xdiff-nodetype"/>
                        <attribute name="parent-xpath" type="string"/>
                    </complexType>
                </element>

                <element name="insert-node-before">
                    <complexType>
                        <sequence>
                            <element name="content" type="anyType"/>
                        </sequence>
                        <attribute name="xpath" type="string"/>
                        <attribute name="node-type" type="xd:xdiff-nodetype"/>

                    </complexType>
                </element>

                <element name="delete-node">
                    <complexType>
                        <attribute name="node-type" type="xd:xdiff-nodetype"/>
                        <attribute name="xpath" type="string"/>
                    </complexType>
                </element>

             </choice>
        </complexType>
    </element>
</schema>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

21
Using the XML SQL Utility

An explanation is given of how to use the Extensible Markup Language (XML) SQL Utility (XSU).

21.1 Introduction to the XML SQL Utility (XSU)

XML SQL Utility (XSU) is an Oracle XML Developer's Kit (XDK) component that lets you transfer XML data using Oracle SQL statements.

You can use XML SQL Utility (XSU) to perform these tasks:

- Transform data in object-relational database tables or views into XML. XSU can query the database and return the result set as an XML document.
- Extract data from an XML document and use canonical mapping to insert the data into a table or a view or update or delete values of the appropriate columns or attributes.

21.1.1 Prerequisites for Using the XML SQL Utility (XSU)

Prerequisites for using the XML SQL Utility (XSU) are covered.

This section assumes that you are familiar with these technologies:

- Oracle Database structured query language (SQL). XSU transfers XML to and from a database through `SELECT` statements and data manipulation language (DML).
- Procedural Language/Structured Query Language (PL/SQL). XDK supplies a PL/SQL application programming interface (API) for XSU that mirrors the Java API.
- Java Database Connectivity (JDBC). Java applications that use XSU to transfer XML to and from a database require a JDBC connection.

21.1.2 XSU Features

The main features provided by XML SQL Utility (XSU) are described.

XSU:

- Dynamically generates document type definitions (DTDs) or XML schemas.
- Generates XML documents in their string or Document Object Model (DOM) representations.
- Performs simple transformations during generation such as modifying default tag names for each `<ROW>` element. You can also register an XSL transformation that XSU applies to the generated XML documents as needed.
- Generates XML as a stream of Simple API for XML (SAX2) callbacks.
- Supports XML attributes during generation, which enables you to specify that a particular column or group of columns maps to an XML attribute instead of an XML element.

Copyright © 1999, 2025, Oracle and/or its affiliates.

- Allows SQL-to-XML-tag escaping. Sometimes column names are not valid XML tag names. To avoid this problem you can either alias all the column names or turn on tag escaping.
- Supports `XMLType` columns in objects or tables.
- Inserts XML into relational database tables or views. When given an XML document, XSU can also update or delete records from a database object.

21.1.3 XSU Restrictions

Some restrictions for using XSU are described.

- XSU can store data only in a single table. You can store XML across tables, however, by using the Oracle Extensible Stylesheet Language Transformation (XSLT) processor to transform a document into multiple documents and inserting them separately. You can also define views over multiple tables and perform insertions into the views. If a view is nonupdatable (because of complex joins), then you can use `INSTEAD OF` triggers over the views to perform the inserts.
- You cannot use XSU to load XML data stored in attributes into a database schema, but you can use an XSLT transformation to change the attributes into elements.
- By default XSU is case-sensitive. You can either use the correct case, or specify that case is to be ignored.
- XSU cannot generate a relational database schema from an input DTD.
- Inserting into `XMLType` tables using XSU is not supported. `XMLType` columns are supported.

21.2 Using the XML SQL Utility: Overview

Topics here include basic XSU use, installing XSU, running the XSU demo programs, and using the XSU command-line utility.

21.2.1 Using XSU: Basic Process

The basic process of using XSU is described.

XSU is accessible through Java classes `OracleXMLQuery` and `OracleXMLSave` in package `oracle.xml.sql.query`. Use class `OracleXMLQuery` to generate XML from relational data and class `OracleXMLSave` to perform DML.

You can write these types of XSU applications:

- Java programs that run inside the database and access the internal XSU Java API
- Java programs that run on the client and access the client-side XSU Java API
- PL/SQL programs that access XSU through PL/SQL packages

21.2.1.1 Generating XML with the XSU Java API: Basic Process

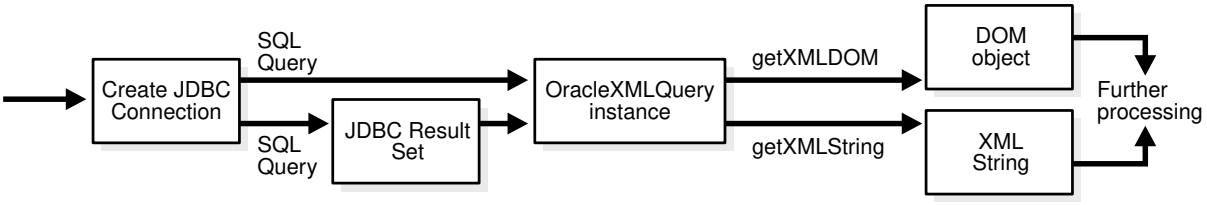
Class `OracleXMLQuery` makes up the XML generation part of the XSU Java API.

Figure 21-1 shows the basic process for generating XML with XSU.

The basic steps in Figure 21-1 are:

Copyright © 1999, 2025, Oracle and/or its affiliates.

Figure 21-1 Generating XML with XSU



1. Create a JDBC connection to the database. Normally, you establish a connection with the `DriverManager` class, which manages a set of JDBC drivers. After the JDBC drivers are loaded, invoke `getConnection()`. When it finds the right driver, this method returns a `Connection` object that represents a database session. All SQL statements are executed within the context of this session.

   You have these options:

   - Create the connection with the JDBC Oracle Call Interface (OCI) driver. This code fragment shows this technique:

     ```
     // import the Oracle driver class
     import oracle.jdbc.*;
     // load the Oracle JDBC driver
     DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
     // create the connection
     Connection conn =
        DriverManager.getConnection("jdbc:oracle:oci:@","hr","password");
     ```

     The preceding example uses the default connection for the JDBC OCI driver.

   - Create the connection with the JDBC thin driver. The thin driver is written in pure Java and can be called from any Java program. This code fragment shows this technique:

     ```
     Connection conn =
        DriverManager.getConnection("jdbc:oracle:thin:@dlsun489:1521:ORCL",
          "hr","password");
     ```

     The thin driver requires the host name (`dlsun489`), port number (1521), and the Oracle system identifier (SID), `ORCL`. The database must have an active Transmission Control Protocol/Internet Protocol (TCP/IP) listener.

   - Use default connection used by the server-side internal JDBC driver. This driver runs within a default session and default transaction context. You are already connected to the database; your SQL operations are part of the default transaction. Thus, you do not have to register the driver. Create the `Connection` object:

     ```
     Connection conn = new oracle.jdbc.OracleDriver().defaultConnection ();
     ```

   > **Note**
   >
   > `OracleXMLDataSetExtJdbc` is used only for Oracle JDBC, whereas `OracleXMLDataSetGenJdbc` is used for non-Oracle JDBC. These classes are in the oracle.xml.sql.dataset package.

2. Create an XML query object and assign it a SQL query. You create an `OracleXMLQuery` Class instance by passing a SQL query to the constructor, as shown in this example:

   ```
   OracleXMLQuery qry = new OracleXMLQuery (conn, "SELECT * from EMPLOYEES");
   ```


Copyright © 1999, 2025, Oracle and/or its affiliates.

3. Configure the XML query object by invoking `OracleXMLQuery` methods. This example specifies that only 20 rows are to be included in the result set:

   ```
   xmlQry.setMaxRows(20);
   ```

4. Return a DOM object or string by invoking `OracleXMLQuery` methods. For example, get a DOM object:

   ```
   XMLDocument domDoc = (XMLDocument)qry.getXMLDOM();
   ```

   Get a string object:

   ```
   String xmlString = qry.getXMLString();
   ```

5. Perform additional processing on the string or DOM as needed.

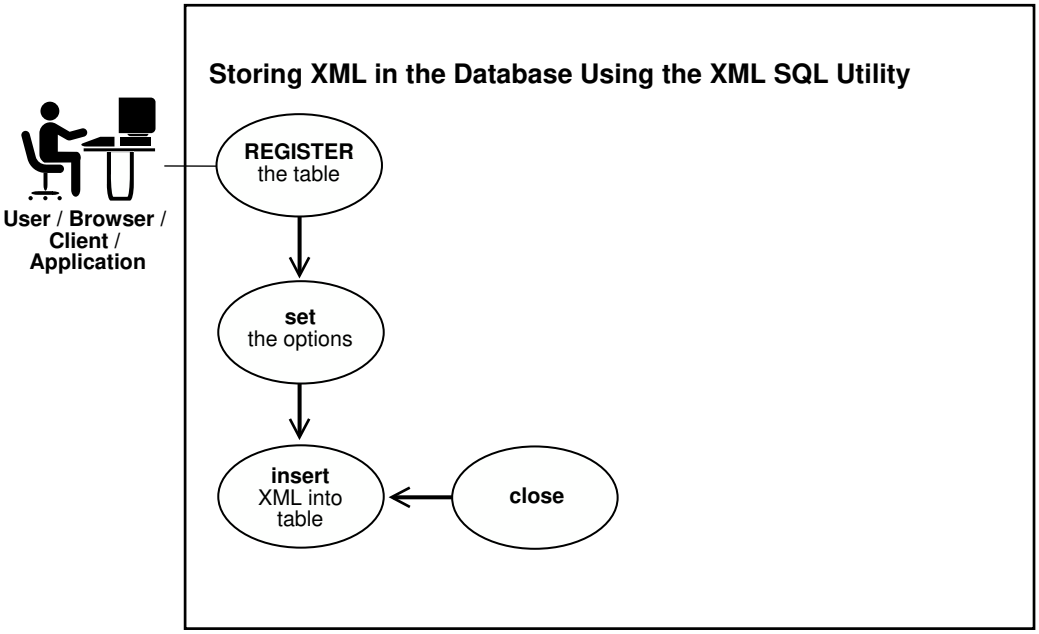
> **See Also**
>
> - *Oracle AI Database Java Developer's Guide* to learn about Oracle JDBC
> - *Oracle AI Database XML Java API Reference* to learn about `OracleXMLQuery` methods

21.2.1.2 Performing DML with the XSU Java API: Basic Process

Use the `OracleXMLSave` class to insert, update, and delete XML in the database.

Figure 21-2 shows the basic process.

Figure 21-2 Storing XML in the Database Using XSU



The basic steps in Figure 21-2 are:

1. Create a JDBC connection to the database. This step is identical to the first step described in Generating XML with the XSU Java API: Basic Process.

Copyright © 1999, 2025, Oracle and/or its affiliates.

2. Create an XML save object and assign it a table on which to perform DML. Pass a table or view name to the constructor, as shown in this example:

```
OracleXMLSave sav = new OracleXMLSave(conn, "employees");
```

3. Specify the primary key columns. For example, this code specifies that `employee_id` is the key column:

```
String [] keyColNames = new String[1];
keyColNames[0] = "EMPLOYEE_ID";
sav.setKeyColumnList(keyColNames);
```

4. Configure the XML save object by invoking `OracleXMLSave` methods. This example specifies an update of the `salary` and `job_id` columns:

```
String[] updateColNames = new String[2];
updateColNames[0] = "SALARY";
updateColNames[1] = "JOB_ID";
sav.setUpdateColumnList(updateColNames); // set the columns to update
```

5. Invoke the `insertXML()`, `updateXML()`, or `deleteXML()` methods on the `OracleXMLSave` object. This example shows an update:

```
// Assume that the user passes in this XML document as the first argument
sav.updateXML(sav.getURL(argv[0]));
```

   When performing the DML, XSU performs these tasks:

   a. Parses the input XML document.
   b. Matches element names to column names in the target table or view.
   c. Converts the elements to SQL types and binds them to the appropriate statement.

6. Close the `OracleXMLSave` object and deallocate all contexts associated with it, as shown in this example:

```
sav.close();
```

> **See Also**
>
> - *Oracle AI Database Java Developer's Guide* to learn about JDBC
> - *Oracle AI Database XML Java API Reference* to learn about `OracleXMLSave`

21.2.2 Installing XSU

XSU is included as part of Oracle Database, along with the other XDK utilities.

XDK for Java Component Dependencies describes the XSU components and dependencies.

By default, the Oracle Universal Installer installs XSU on disk and loads it into the database. No user intervention is required. If you did not load XSU in the database when installing Oracle, you can install XSU manually as follows:

1. Ensure that Oracle XML DB is installed (it is installed by default as part of Oracle Database).
2. Load the `xsu12.jar` file into the database. This JAR file, which has a dependency on `xdb.jar` for `XMLType` access, is described in Table 11-1.
3. Run the `$ORACLE_HOME/rdbms/admin/dbmsxsu.sql` script. This SQL script builds the XSU PL/SQL API.

Copyright © 1999, 2025, Oracle and/or its affiliates.

As explained in Using XSU: Basic Process, you do not have to load XSU into the database to use it. XSU can reside in any tier that supports Java.

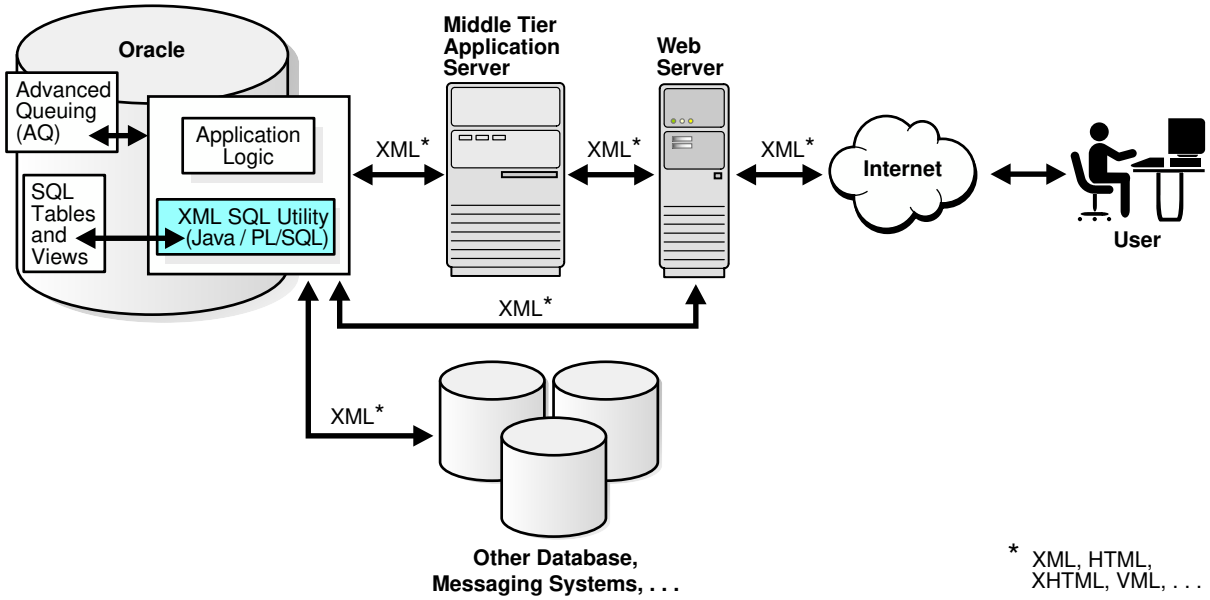
21.2.2.1 XSU in the Database

The typical architecture is shown for applications that use the XSU libraries installed in the database.

Figure 21-3 illustrates this typical architecture. 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. You deliver the XML internally through stored procedures in the database or externally through web servers or application servers.

In Figure 21-3 all lines are bidirectional. Because XSU can generate and save data, resources can deliver XML to XSU running inside the database, which can then insert it in the appropriate database tables.

Figure 21-3 Running XSU in the Database



21.2.2.2 XSU in an Application Server

You can run XSU in an application server.

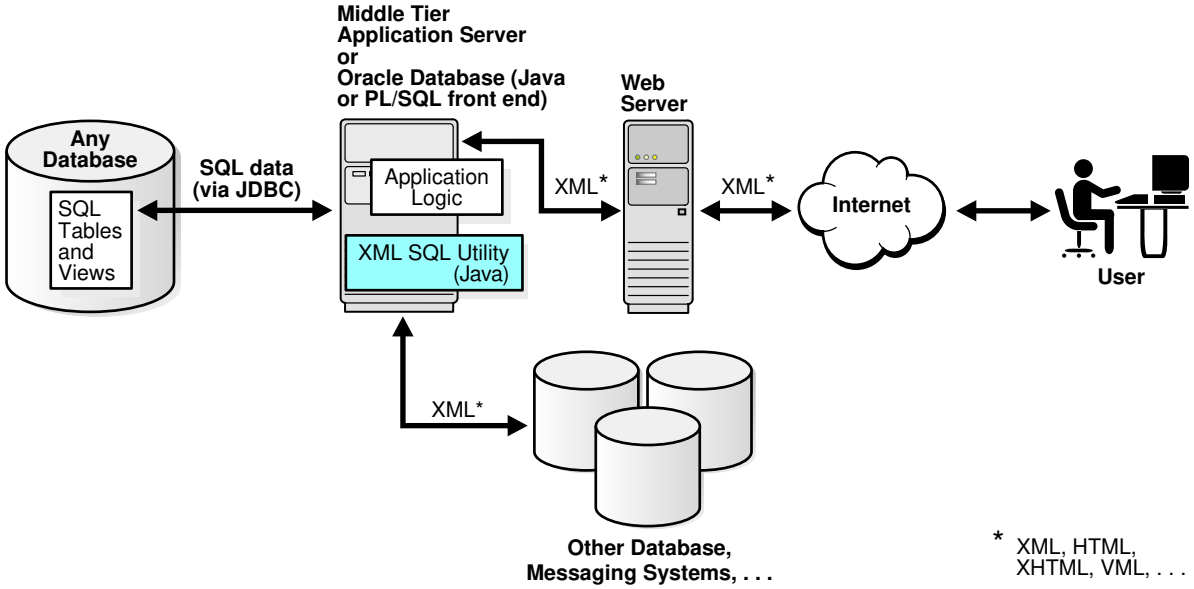
Your application architecture may require an application server in the middle tier. The application tier can be a database or an application server that supports Java programs.

You can 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 can install XSU in your middle tier, thereby enabling your Java programs to make use of XSU through its Java API.

Figure 21-4 shows 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 process is bidirectional, which means that the data can be put back into the JDBC sources (database tables or views) with XSU. If a database is used as the application server, then you can use the PL/SQL front end instead of Java.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Figure 21-4 Running XSU in the Middle Tier

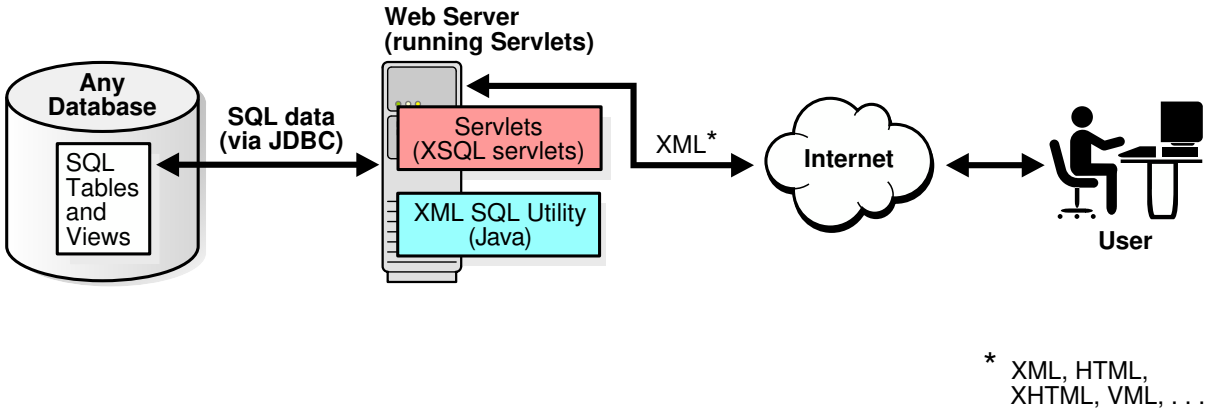


21.2.2.3 XSU in a Web Server

You can run XSU in a web server because the web server supports Java servlets.

Figure 21-5 shows XSU running in a web server.

Figure 21-5 Running XSU in a Web Server



You can write Java servlets that use XSU. 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. To perform XML processing in the web server and avoid intricate servlet programming, you can use the XSQL Servlet.

Copyright © 1999, 2025, Oracle and/or its affiliates.

See Also

- *Oracle XML DB Developer's Guide*, especially the chapter on generating XML, for examples on using XSU with `XMLType`
- *Oracle AI Database XML Java API Reference* to learn about the classes `OracleXMLQuery` and `OracleXMLSave`
- Using the XSQL Pages Publishing Framework to learn about XSQL Servlet

21.2.3 Running the XSU Demo Programs

Demo programs for XSU are included in `$ORACLE_HOME/xdk/demo/java/xsu`.

Table 21-1 describes the XML files and programs that you can use to test XSU.

Table 21-1 XSU Sample Files

File	Description
`createObjRelSchema.sql`	A SQL script that sets up an object-relational schema and populates it. See XML Mapping Against an Object-Relational Schema.
`createObjRelSchema2.sql`	A SQL script that sets up an object-relational schema and populates it. See Altering the Database Schema or SQL Query.
`createRelSchema.sql`	A SQL script that creates a relational table and then creates a customer view that contains a customer object on top of it. See Altering the Database Schema or SQL Query.
`customer.xml`	An XML document that describes a customer. See Altering the Database Schema or SQL Query.
`domTest.java`	A program that generates a DOM tree and then traverses it in document order, printing the nodes one by one. See Generating a DOM Tree with OracleXMLQuery.
`index.txt`	A README that describes the programs in the demo directory.
`mapColumnToAtt.sql`	A SQL script that queries the `employees` table, rendering `employee_id` as an XML attribute. See Altering the Database Schema or SQL Query.
`new_emp.xml`	An XML document that describes a new employee. See Running the testInsert Program.
`new_emp2.xml`	An XML document that describes a new employee. See Running the testInsertSubset Program.
`noRowsTest.java`	A program that throws an exception when there are no more rows. See Raising a No Rows Exception.
`pageTest.java`	A program that uses the JDBC `ResultSet` to generate XML one page at a time. See Generating Scrollable Result Sets.
`paginateResults.java`	A program that generates an XML page that paginates results. See Paginating Results with OracleXMLQuery: Example.
`refCurTest.java`	A program that generates XML from the results of the SQL query defined in the `testRefCur` function. See Generating XML from Cursor Objects.
`samp1.java`	A program that queries the `scott.emp` table, then generates an XML document from the query results.
`samp10.java`	A program that inserts `sampdoc.xml` into the `xmltest_tab1` table.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 21-1 (Cont.) XSU Sample Files

File	Description
samp2.java	A program that queries the scott.emp table, then generates an XML document from the query results. This program demonstrates how you can customize the generated XML document.
sampdoc.xml	A sample XML data document that samp10.java inserts into the database.
samps.sql	A SQL script that creates the xmltest_tab1 table used by samp10.java.
testDeleteKey.java	A program that limits the number of elements used to identify a row, which improves performance by caching the DELETE statement and batching transactions. See Deleting by Key with OracleXMLSave.
testDeleteRow.java	A program that accepts an XML document file name as input and deletes the rows corresponding to the elements in the document. See Deleting by Row with OracleXMLSave.
testException.java	A sample program shown that throws a runtime exception and then gets the parent exception by invoking Exception.getParentException(). See Getting the Parent Exception.
testInsert.java	A Java program that inserts XML values into all columns of the hr.employees table. See Inserting XML into All Columns with OracleXMLSave.
testInsertSubset.java	A program shown that inserts XML data into a subset of columns. See Inserting XML into a Subset of Columns with OracleXMLSave.
testRef.sql	A PL/SQL script that creates a function that defines a REF cursor and returns it. Every time the testRefCur function is called, it opens a cursor object for the SELECT query and returns that cursor instance. See Generating XML from Cursor Objects.
testUpdate.java	A sample program that updates the hr.employees table by invoking the OracleXMLSave.setKeyColumnList() method. See Updating Rows Using OracleXMLSave.
testUpdateList.java	Suppose you want to update only the salary and job title for each employee and ignore the other information. If you know that all the elements to be updated are the same for all ROW elements in the XML document, then you can use the OracleXMLSave.setUpdateColumnNames() method to specify the columns. See Updating a Column List Using OracleXMLSave.
testXMLSQL.java	A sample program that uses XSU to generate XML as a String object. This program queries the hr.employees table and prints the result set to standard output. See Generating a String with OracleXMLQuery.
upd_emp.xml	An XML document that contains updated salary and other information for a series of employees. See Running the testUpdate Program.
upd_emp2.xml	An XML document that contains updated salary and other information for a series of employees. See Running the testUpdate Program.
updateEmployee.sql	An XML document that contains new data for two employees. See Running the testUpdateList Program.

The steps for running the demos are:

1. Change into the $ORACLE_HOME/xdk/demo/java/xsu directory (UNIX) or %ORACLE_HOME%\xdk\demo\java\xsu directory (Windows).
2. Ensure that your environment variables are set as described in Setting Up the XDK for Java Environment. In particular, ensure that the Java classpath includes xsu12.jar for XSU and ojdbc6.jar (Java 1.6) for JDBC. If you use a multibyte character set other than

Copyright © 1999, 2025, Oracle and/or its affiliates.

UTF-8, ISO8859-1, or JA16SJIS, then place `orai18n.jar` in your classpath so that JDBC can convert the character set of the input file to the database character set.

3. Compile the Java programs as shown in this example:

```
javac samp1.java samp2.java samp10.java
```

4. Connect to a database as user `hr` and run SQL script `createRelSchema`:

```
CONNECT hr
@$ORACLE_HOME/xdk/demo/java/xsu/createRelSchema
```

These sections describe the XSU demos in detail.

21.2.4 Using the XSU Command-Line Utility

XDK includes a command-line Java interface for XSU. XSU command-line options are provided through the Java class `OracleXML`.

To use this API ensure that your Java classpath is set as described in Setting Up the XDK for Java Environment.

To print usage information for XSU to standard output, run this command:

```
java OracleXML
```

To use XSU, invoke it with either the `getXML` or `putXML` parameter:

```
java OracleXML getXML options
java OracleXML putXML options
```

Table 21-2 describes the `getXML` options.

Table 21-2 getXML Options

getXML Option	Description
`-user "username/password"`	Specifies the user name and password to connect to the database. The connect string is also specified. You can specify the user name and password as part of the connect string.
`-conn "JDBC_connect_string"`	Specifies the JDBC database connect string. By default the connect string is: "`jdbc:oracle:oci:@`".
`-withDTD`	Instructs the XSU to generate the DTD along with the XML document.
`-withSchema`	Instructs the XSU to generate the schema along with the XML document.
`-rowsetTag tag_name`	Specifies the rowset tag, which is tag that encloses all the XML elements corresponding to the records returned by the query. The default rowset tag is <ROWSET>. If you specify an empty string ("") for rowset, then XSU omits the rowset element.
`-rowTag tag_name`	Specifies the row tag that encloses the data corresponding to a database row. The default row tag is <ROW>. If you specify an empty string ("") for the row tag, then XSU omits the row tag.
`-rowIdAttr row_id_attribute_name`	Names the attribute of the `ROW` element that keeps track of the cardinality of the `rows`. By default this attribute is `num`. If you specify an empty string as the `rowID` attribute, then XSU omits the attribute.
`-rowIdColumn row_Id_column_name`	Specifies that the value of a scalar column from the query is to be used as the value of the `rowID` attribute.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 21-2 (Cont.) getXML Options

getXML Option	Description
`-collectionIdAttr` *collect_id_attr_name*	Names the attribute of an XML list element that keeps track of the cardinality of the elements of the list. The generated XML lists correspond to either a cursor query, or collection. If you specify an empty string ("") as the `rowID` attribute, then XSU omits the attribute.
`-useTypeForCollElemTag`	Specifies the use type name for the column-element tag. By default XSU uses the `column-name_item`.
`-useNullAttrId`	Specifies the attribute `NULL (TRUE/FALSE)` to indicate the nullness of an element.
`-styleSheet` *stylesheet_URI*	Specifies the stylesheet in the XML processing instruction.
`-stylesheetType` *stylesheet_type*	Specifies the stylesheet type in the XML processing instruction.
`-setXSLT` *URI*	Specifies the XSLT stylesheet to apply to the XML document.
`-setXSLTRef` *URI*	Sets the XSLT external entity reference.
`-useLowerCase | -useUpperCase`	Generates lowercase or uppercase tag names. The default is to match the case of the SQL object names from which the tags are generated.
`-withEscaping`	Specifies the treatment of characters that are legal in SQL object names but illegal in XML tags. If such a character is encountered, then it is escaped so that it does not throw an exception.
`-errorTag` *error tag_name*	Specifies the tag to enclose error messages that are formatted as XML.
`-raiseException`	Specifies that XSU must throw a Java exception. By default XSU catches any error and produces the XML error.
`-raiseNoRowsException`	Raises an exception if no rows are returned.
`-useStrictLegalXMLCharCheck`	Performs strict checking on input data.
`-maxRows` *maximum_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 that contains the query or the query itself.

Table 21-3 describes the `putXML` options.

Table 21-3 putXML Options

putXML Options	Description
`-user "`*username*`/`*password*`"`	Specifies the user name and password to connect to the database. The connect string is also specified. You can specify the user name and password as part of the connect string.
`-conn "`*JDBC_connect_string*`"`	Specifies the JDBC database connect string. By default the connect string is: `"jdbc:oracle:oci:@"`.
`-batchSize` *batching_size*	Specifies the batch size that controls the number of rows that are batched together and inserted in a single trip to the database to improve performance.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 21-3 (Cont.) putXML Options

putXML Options	Description
`-commitBatch` *`commit_size`*	Specifies the number of inserted records after which a commit is to be executed. If the autocommit is `TRUE` (the default), then setting `commitBatch` has no consequence.
`-rowTag` *`tag_name`*	Specifies the `row` tag, which is tag used to enclose the data corresponding to a database row. The default row tag is `<ROW>`. If you specify an empty string for the row tag, then XSU omits the row tag.
`-dateFormat` *`date_format`*	Specifies the date format for the date values in the XML document.
`-withEscaping`	Turns on reverse mapping if SQL to XML name escaping was used when generating the doc.
`-ignoreCase`	Makes the matching of the column names with tag names case insensitive. For example, `EmpNo` matches with `EMPNO` if `ignoreCase` is on.
`-preserveWhitespace`	Preserves the white space in the inserted XML document.
`-setXSLT` *`URI`*	Specifies the XSLT to apply to the XML document before inserting.
`-setXSLTRef` *`URI`*	Sets the XSLT external entity reference.
`-fileName` *`file_name`* | `-URL` *`URL`* | `-xmlDoc` *`xml_document`*	Specifies the XML document to insert: a local file, a URL, or an XML document as a string on the command line.
`table_name`	Specifies the name of the table to put the values into.

21.2.4.1 Generating XML with the XSU Command-Line Utility

To generate XML from the database schema use the `getXML` parameter.

For example, to generate an XML document by querying the `employees` table in the `hr` schema, you can use this syntax:

```
java OracleXML getXML -user "hr/password" "SELECT * FROM employees"
```

The preceding command performs these tasks:

1. Connects to the current default database
2. Executes the specified `SELECT` query
3. Converts the SQL result set to XML
4. Prints the XML to standard output

The `getXML` parameter supports a wide range of options, which are explained in Table 21-2.

21.2.4.2 Generating XMLType Data with the XSU Command-Line Utility

You can use XSU to generate XML from tables with `XMLType` columns.

Suppose that you run the demo script `setup_xmltype.sql` to create and populate the `parts` table. You can generate XML from this table with XSU:

```
java OracleXML getXML -user "hr/password" -rowTag "Part" "SELECT * FROM parts"
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

The output of the command is shown below:

```
<?xml version = '1.0'?>
<ROWSET>
   <Part num="1">
      <PARTNO>1735</PARTNO>
      <PARTNAME>Gizmo</PARTNAME>
      <PARTDESC>
         <Description>
           <Title>Description of the Gizmo</Title>
           <Author>John Smith</Author>
           <Body>
             The <b>Gizmo</b> is <i>grand</i>.
           </Body>
         </Description>
      </PARTDESC>
   </Part>
</ROWSET>
```

21.2.4.3 Performing DML with the XSU Command-Line Utility

An example shows how to insert an XML document into a database table.

To insert an XML document called `new_employees.xml` into the `hr.employees` table, use this syntax:

```
java OracleXML putXML -user "hr/password" -fileName "new_employees.xml" employees
```

The preceding command performs these tasks:

1. Connects to the current database as `hr`
2. Reads the XML document named `new_emp.xml`
3. Parses the XML document, matching the tags with column names
4. Inserts the values appropriately into the `employees` table

The `getXML` parameter supports a wide range of options, which are explained in Table 21-2.

21.3 Programming with the XSU Java API

Topics here include using OracleXMLQuery and OracleXMLSave to perform various operations, and handling XSU Java exceptions.

21.3.1 Generating a String with OracleXMLQuery

The `testXMLSQL.java` demo program uses XSU to generate XML as a `String` object. The program queries table `hr.employees` and prints the result set to standard output.

The `testXMLSQL.java` program follows these steps:

1. Register the JDBC driver and create a database connection. This code fragment uses the OCI JDBC driver and connects with the user name `hr`:

   ```
   import oracle.jdbc.*;...Connection conn  = getConnection("hr","password");
   ...
   private static Connection getConnection(String username, String password)
   ```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
    throws SQLException
{
// register the JDBC driver
  DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
// create the connection using the OCI driver
  Connection conn =
    DriverManager.getConnection("jdbc:oracle:oci:@",username,password);
  return conn;
}
```

2. Create an XML query object and initialize it with a SQL query. This code fragment initializes the object with a `SELECT` statement on `hr.employees`:

   ```
   OracleXMLQuery qry = new OracleXMLQuery(conn, "SELECT * FROM employees");
   ```

3. Get the query result set as a `String` object. The `getXMLString()` method transforms the object-relational data specified in the constructor into an XML document. This example shows this technique:

   ```
   String str = qry.getXMLString();
   ```

4. Close the query object to release any resources, as shown in this code:

   ```
   qry.close();
   ```

21.3.1.1 Running the testXMLSQL Program

The `testXMLSQL` program is described.

To run the `testXMLSQL.java` program perform these steps:

1. Compile `testXMLSQL.java` with `javac`.
2. Execute `java testXMLSQL` on the command line.

You must have the `CLASSPATH` pointing to this directory for the Java executable to find the class. Alternatively, use visual Java tools such as Oracle JDeveloper to compile and run this program. When run, this program prints out the XML file to the screen. This code shows sample output with some rows edited out:

```
<?xml version = '1.0'?>
<ROWSET>
   <ROW num="1">
      <EMPLOYEE_ID>100</EMPLOYEE_ID>
      <FIRST_NAME>Steven</FIRST_NAME>
      <LAST_NAME>King</LAST_NAME>
      <EMAIL>SKING</EMAIL>
      <PHONE_NUMBER>515.123.4567</PHONE_NUMBER>
      <HIRE_DATE>6/17/1987 0:0:0</HIRE_DATE>
      <JOB_ID>AD_PRES</JOB_ID>
      <SALARY>24000</SALARY>
      <DEPARTMENT_ID>90</DEPARTMENT_ID>
   </ROW>
<!-- ROW num="2" through num="107" ... -->
</ROWSET>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

21.3.2 Generating a DOM Tree with OracleXMLQuery

To generate a DOM tree from the XML generated by XSU, you can directly request a DOM document from XSU. This technique saves the overhead of creating a string representation of the XML document and then parsing it to generate the DOM tree.

XSU invokes the Oracle XML parser to construct the DOM tree from the data values. The `domTest.java` demo program generates a DOM tree and then traverses it in document order, printing the nodes one by one.

The first two steps in the `domTest.java` program are the same as in the `testXMLSQL.java` program described in Generating a String with OracleXMLQuery. The program proceeds as follows:

1. Get the DOM by invoking `getXMLDOM()` method. The following example shows this technique:

   ```
   XMLDocument domDoc = (XMLDocument)qry.getXMLDOM();
   ```

2. Print the DOM tree. The following code prints to standard output:

   ```
   domDoc.print(System.out);
   ```

   You can also create a `StringWriter` and wrap it in a `PrintWriter`:

   ```
   StringWriter s = new StringWriter(10000);
   domDoc.print(new PrintWriter(s));
   System.out.println(" The string version ---> \n"+s.toString());
   ```

After compiling the program, run it from the command line:

```
java domTest
```

21.3.3 Paginating Results with OracleXMLQuery

Topics here include limiting the rows in a result set, keeping an object open during a user session, and paginating results using `OracleXMLQuery`.

21.3.3.1 Limiting the Number of Rows in the Result Set

Different ways to limit the number of rows in a result set are described.

In `testXMLSQL.java` and `domTest.java`, XSU generated XML from all rows returned by the query. Suppose that you query a table that contains 1000 rows, but you want only 100 rows at a time. One approach is to execute one query to get the first 100 rows, another to get the next 100 rows, and so on. With this technique you cannot skip the first five rows of the query and then generate the result. To avoid these problems, use these Java methods:

- `OracleXMLSave.setSkipRows()` forces XSU to skip the desired number of rows before starting to generate the result. The command-line equivalent to this method is the `-skipRows` parameter.
- `OracleXMLSave.setMaxRows()` limits the number of rows converted to XML. The command-line equivalent to this method is the `-maxRows` parameter.

Example 21-1 sets `skipRows` to a value of `5` and `maxRows` to a value of `1`, which causes XSU to skip the first 5 rows and then generate XML for the next row when querying the `hr.employees` table.

Copyright © 1999, 2025, Oracle and/or its affiliates.

The following shows sample output (only row 6 of the query result set is returned):

```
<?xml version = '1.0'?>
<ROWSET>
   <ROW num="6">
      <EMPLOYEE_ID>105</EMPLOYEE_ID>
      <FIRST_NAME>David</FIRST_NAME>
      <LAST_NAME>Austin</LAST_NAME>
      <EMAIL>DAUSTIN</EMAIL>
      <PHONE_NUMBER>590.423.4569</PHONE_NUMBER>
      <HIRE_DATE>6/25/1997 0:0:0</HIRE_DATE>
      <JOB_ID>IT_PROG</JOB_ID>
      <SALARY>4800</SALARY>
      <MANAGER_ID>103</MANAGER_ID>
      <DEPARTMENT_ID>60</DEPARTMENT_ID>
   </ROW>
</ROWSET>
```

Example 21-1 Specifying skipRows and maxRows on the Command Line

```
java OracleXML getXML -user "hr/password" -skipRows 5 -maxRows 1 \
  "SELECT * FROM employees"
```

21.3.3.2 Keeping an Object Open for the Duration of the User's Session

In some situations, you might want to keep a query object open for the duration of the user session. You can handle such cases with the `maxRows()` method and the `keepObjectOpen()` method.

Consider a web search engine that paginates search results. The first page lists 10 results, the next page lists 10 more, and so on. To perform this task with XSU, request 10 rows at a time and keep the `ResultSet` open so that the next time you ask XSU for more results, it starts generating from where the last generation finished. If `OracleXMLQuery` creates a result set from the SQL query string, then it typically closes the `ResultSet` internally because it assumes no more results are required. Thus, you must invoke `keepObjectOpen()` to keep the cursor active.

A different case requiring an open query object is when the number of rows or number of columns in a row is very large. In this case, you can generate multiple small documents rather than one large document.

Related Topics

- Paginating Results with OracleXMLQuery: Example
  The `paginateResults.java` program shows how you can generate an XML page that paginates results. The output XML displays only 20 rows of the `hr` table.

21.3.3.3 Paginating Results with OracleXMLQuery: Example

The `paginateResults.java` program shows how you can generate an XML page that paginates results. The output XML displays only 20 rows of the `hr` table.

The `paginateResults.java` program shows how you can generate an XML page that paginates results. The output XML displays only 20 rows of the `hr` table.

The first step of the `paginateResults.java` program, which creates the connection, is the same as in `testXMLSQL.java`. The program continues as follows:

1. Create a SQL statement object and initialize it with a SQL query. The following code fragment sets two options in `java.sql.ResultSet`:

Copyright © 1999, 2025, Oracle and/or its affiliates.

```
Statement stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                      ResultSet.CONCUR_READ_ONLY);
```

2. Create the query as a string and execute it by invoking `Statement.executeQuery()`. The return object is of type `ResultSet`. The following example shows this technique:

```
String sCmd = "SELECT first_name, last_name FROM hr.employees";
ResultSet rs = stmt.executeQuery(sCmd);
```

3. Create the query object, as shown in this code:

```
OracleXMLQuery xmlQry = new OracleXMLQuery(conn, rs);
```

4. Configure the query object. The following code specifies that the query object is open for the duration of the session. It also limits the number of rows returned to 20:

```
xmlQry.keepObjectOpen(true);
xmlQry.setRowsetTag("ROWSET");
xmlQry.setRowTag("ROW");
xmlQry.setMaxRows(20);
```

5. Retrieve the result as a `String` and print:

```
String sXML = xmlQry.getXMLString();
System.out.println(sXML);
```

After compiling the program, run it from the command line:

```
java paginateResults
```

21.3.4 Generating Scrollable Result Sets

You might want to perform a query and then retrieve a previous page of results from within the result set. To enable scrolling, instantiate the `Oracle.jdbc.ResultSet` class. You can use the `ResultSet` object to move back and forth within the result set and use XSU to generate XML each time.

The `pageTest.java` program shows how to use the JDBC `ResultSet` to generate XML a page at a time. Using `ResultSet` may be necessary in cases that are not handled directly by XSU, for example, when setting the batch size and binding values.

The `pageTest.java` program creates a `pageTest` object and initializes it with a SQL query. The constructor for the `pageTest` object performs these steps:

1. Create a JDBC connection by invoking the same `getConnection()` method defined in `paginateResults.java`:

```
Connection conn;
...
conn  = getConnection("hr","password");
```

2. Create a statement:

```
Statement stmt;
...
stmt = conn.createStatement();
```

3. Execute the query passed to the constructor to get the scrollable result set. The following code shows this technique:

```
ResultSet rset = stmt.executeQuery(sqlQuery);
```

4. Create a query object by passing references to the connection and result set objects to the constructor. The following code fragment shows this technique:

Copyright © 1999, 2025, Oracle and/or its affiliates.

```
OracleXMLQuery qry;
...
qry = new OracleXMLQuery(conn,rset);
```

5. Configure the query object. The following code fragment specifies that the query object be kept open, and that it raise an exception when there are no more rows:

```
qry.keepObjectOpen(true);
qry.setRaiseNoRowsException(true);
qry.setRaiseException(true);
```

6. After creating the query object by passing it the string `"SELECT * FROM employees"`, the program loops through the result set. The `getResult()` method receives integer values specifying the start row and end row of the set. It sets the maximum number of rows to retrieve by calculating the difference of these values and then retrieves the result as a string. The following `while` loop retrieves and prints ten rows at a time:

```
int i = 0;
while ((str = test.getResult(i,i+10))!= null)
{
  System.out.println(str);
  i+= 10;
}
```

After compiling the program, run it from the command line:

```
java pageTest
```

21.3.5 Generating XML from Cursor Objects

You can initialize a `CallableStatement` object, execute a PL/SQL function that returns a cursor variable, get the `OracleResultSet` object, and send it to an `OracleXMLQuery` object to obtain the desired XML data.

Class `OracleXMLQuery` provides XML conversion only for query strings or `ResultSet` objects. If your program uses PL/SQL procedures that return `REF` cursors, then how do you perform the conversion? You can use the `ResultSet` conversion mechanism described in Generating Scrollable Result Sets.

`REF` cursors are references to cursor objects in PL/SQL. These cursor objects are SQL statements over which a program can iterate to get a set of values. The cursor objects are converted into `OracleResultSet` objects in the Java world. In your Java program you can initialize a `CallableStatement` object, execute a PL/SQL function that returns a cursor variable, get the `OracleResultSet` object, and then send it to the `OracleXMLQuery` object to get the desired XML.

Consider the `testRef` PL/SQL package defined in the `testRef.sql` script. It creates a function that defines a `REF` cursor and returns it. Every time the `testRefCur` PL/SQL function is called, it opens a cursor object for the `SELECT` query and returns that cursor instance. To convert the object to XML, do this:

1. Run the `testRef.sql` script to create the `testRef` package in the `hr` schema.
2. Compile and run the `refCurTest.java` program to generate XML from the results of the SQL query defined in the `testRefCur` function.

To apply the stylesheet, you can use the `applyStylesheet` command, which forces the stylesheet to be applied before generating the output.

Copyright © 1999, 2025, Oracle and/or its affiliates.

21.3.6 Inserting Rows with OracleXMLSave

To insert a document into a table or view, supply the table or view name and the document. XSU parses the document and creates an `INSERT` statement into which it binds the values. By default, XSU inserts values into all columns of the table or view.

An absent element is treated as a `NULL` value. The following example shows how you can store the XML document generated from the `hr.employees` table in the table.

21.3.6.1 Inserting XML into All Columns with OracleXMLSave

The `testInsert.java` demo program inserts XML values into all columns of the `hr.employees` table.

The program follows these steps:

1. Create a JDBC OCI connection. The program invokes the same `getConnection()` method used by the previous examples in this chapter:

   ```
   Connection conn = getConnection("hr","password");
   ```

2. Create an XML save object. You initialize the object by passing it the `Connection` reference and the name of the table on which you want to perform DML. The following example shows this technique:

   ```
   OracleXMLSave sav = new OracleXMLSave(conn, "employees");
   ```

3. Insert the data in an input XML document into the `hr.employees` table. The following code fragment creates a URL from the document file name specified on the command line:

   ```
   sav.insertXML(sav.getURL(argv[0]));
   ```

4. Close the XML save object:

   ```
   sav.close();
   ```

21.3.6.1.1 Running the testInsert Program

The `textInsert` program is described.

Assume that you write the `new_emp.xml` document to describe new employee Janet Smith, who has employee ID 7369. You pass the file name `new_emp.xml` as an argument to the `testInsert` program:

```
java testInsert "new_emp.xml"
```

The program inserts a new row in the `employees` table that contains the values for the columns specified. Any absent element inside the row element is treated as `NULL`.

Running the program generates an `INSERT` statement of this form:

```
INSERT INTO hr.employees
  (employee_id, first_name, last_name, email, phone_number, hire_date,
   salary, commission_pct, manager_id, department_id)
VALUES
  (?, ?, ?, ?, ?, ?, ?, ?, ?, ?);
```

XSU matches the element tags in the input XML document that match the column names and binds their values.

Copyright © 1999, 2025, Oracle and/or its affiliates.

21.3.6.2 Inserting XML into a Subset of Columns with OracleXMLSave

In some situations, you might not want to insert values into all columns. For example, the group of values that you get might not be the complete set, requiring you to use triggers or default values for the remaining columns.

The `testInsertSubset.java` demo program shows how to handle this case. It follows these steps:

1. Create a JDBC OCI connection. The program invokes the same `getConnection()` method used by the previous examples in this chapter:

   ```
   Connection conn = getConnection("hr","password");
   ```

2. Create an XML save object. Initialize the object by passing it the `Connection` reference and the name of the table on which you want to perform DML. The following example shows this technique:

   ```
   OracleXMLSave sav = new OracleXMLSave(conn, "employees");
   ```

3. Create an array of strings. Each element of the array must contain the name of a column in which values are inserted. The following code fragment specifies the names of five columns:

   ```
   String [] colNames = new String[5];
   colNames[0] = "EMPLOYEE_ID";
   colNames[1] = "LAST_NAME";
   colNames[2] = "EMAIL";
   colNames[3] = "JOB_ID";
   colNames[4] = "HIRE_DATE";
   ```

4. Configure the XML save object to update the specified columns. The following statement passes a reference to the array to the `OracleXMLSave.setUpdateColumnList()` method:

   ```
   sav.setUpdateColumnList(colNames);
   ```

5. Insert the data in an input XML document into the `hr.employees` table. The following code fragment creates a URL from the document file name specified on the command line:

   ```
   sav.insertXML(sav.getURL(argv[0]));
   ```

6. Close the XML save object:

   ```
   sav.close();
   ```

21.3.6.2.1 Running the testInsertSubset Program

The `testInsertSubset` program is described.

Assume that you use the `new_emp2.xml` document to store data for new employee Adams, who has employee ID 7400. You pass `new_emp2.xml` as an argument to the `testInsert` program:

```
java testInsert new_emp2.xml
```

The program ignores values for the columns that were not specified in the input file. It performs an `INSERT` for each `ROW` element in the input and batches the `INSERT` statements by default.

The program generates this `INSERT` statement:

```
INSERT INTO hr.employees (employee_id, last_name, email, job_id, hire_date)
  VALUES (?, ?, ?, ?, ?);
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

21.3.7 Updating Rows Using OracleXMLSave

Examples show how to update the fields in a table or view. You supply the table or view name and an XML document. XSU parses the document (if a string is given) and creates one or more `UPDATE` statements into which it binds all of the values.

The following examples use an XML document to update table `hr.employees`.

21.3.7.1 Updating Key Columns Using OracleXMLSave

Demo program `testUpdate.java` invokes method `OracleXMLSave.setKeyColumnList()` to update table `hr.employees`.

`testUpdate.java` follows these steps:

1. Create a JDBC OCI connection. The program invokes the same `getConnection()` method used by the previous examples in this chapter:

   ```
   Connection conn = getConnection("hr","password");
   ```

2. Create an XML save object. You initialize the object by passing it the `Connection` reference and the name of the table on which you want to perform DML. The following example shows this technique:

   ```
   OracleXMLSave sav = new OracleXMLSave(conn, "employees");
   ```

3. Create a single-element `String` array to hold the name of the primary key column in the table to be updated. The following code fragment specifies the name of the `employee_id` column:

   ```
   String [] keyColNames = new String[1];
   colNames[0] = "EMPLOYEE_ID";
   ```

4. Set the XML save object to the primary key specified in the array. The following statement passes a reference to the `keyColNames` array to the `OracleXMLSave.setKeyColumnList()` method:

   ```
   sav.setKeyColumnList(keyColNames);
   ```

5. Update the rows specified in the input XML document. The following statement creates a URL from the file name specified on the command line:

   ```
   sav.updateXML(sav.getURL(argv[0]));
   ```

6. Close the XML save object:

   ```
   sav.close();
   ```

21.3.7.1.1 Running the testUpdate Program

The `testUpdate` program is described.

You can use XSU to update specified fields in a table. Example 21-2 shows `upd_emp.xml`, which contains updated salary and other information for the two employees that you just added, 7369 and 7400.

For updates, supply XSU with the list of key column names in the `WHERE` clause of the `UPDATE` statement. In the `hr.employees` table the `employee_id` column is the key.

Pass the file name `upd_emp.xml` as an argument to the preceding program:

```
java testUpdate upd_emp.xml
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

The program generates two UPDATE statements. For the first ROW element, the program generates an UPDATE statement to update the SALARY field:

```
UPDATE hr.employees SET salary = 3250 WHERE employee_id = 7400;
```

For the second ROW element the program generates this statement:

```
UPDATE hr.employees SET job_id = 'SA_REP' AND MANAGER_ID = 145
  WHERE employee_id = 7369;
```

Example 21-2 upd_emp.xml

```
<?xml version='1.0'?>
<ROWSET>
 <ROW num="1">
    <EMPLOYEE_ID>7400</EMPLOYEE_ID>
    <SALARY>3250</SALARY>
 </ROW>
 <ROW num="2">
    <EMPLOYEE_ID>7369</EMPLOYEE_ID>
    <JOB_ID>SA_REP</JOB_ID>
    <MANAGER_ID>145</MANAGER_ID>
 </ROW>
<!-- additional rows ... -->
</ROWSET>
```

21.3.7.2 Updating a Column List Using OracleXMLSave

You can update a table using only a subset of the elements in an XML document, by specifying a list of columns. This is fast because XSU uses the same UPDATE statement, with bind variables for all of the ROW elements. Other tags in the document can be ignored.

> **Note**
>
> When you specify a list of columns to update, if an element corresponding to an update column is absent, XSU treats it as NULL.

Suppose you want to update the salary and job title for each employee and ignore the other data. If you know that all the elements to be updated are the same for all ROW elements in the XML document, then you can use the OracleXMLSave.setUpdateColumnNames() method to specify the columns. The testUpdateList.java program shows this technique.

The testUpdateList.java program follows these steps:

1. Create a JDBC OCI connection. The program invokes the same getConnection() method used by the previous examples in this chapter:

   ```
   Connection conn = getConnection("hr","password");
   ```

2. Create an XML save object. You initialize the object by passing it the Connection reference and the name of the table on which you want to perform DML. The following example shows this technique:

   ```
   OracleXMLSave sav = new OracleXMLSave(conn, "employees");
   ```

3. Create an array of type String to hold the name of the primary key column in the table to be updated. The array contains only one element, which is the name of the primary key

Copyright © 1999, 2025, Oracle and/or its affiliates.

column in the table to be updated. The following code fragment specifies the name of the `employee_id` column:

```
String [] colNames = new String[1];
colNames[0] = "EMPLOYEE_ID";
```

4. Set the XML save object to the primary key specified in the array. The following statement passes a reference to the `colNames` array to the `OracleXMLSave.setKeyColumnList()` method:

```
sav.setKeyColumnList(keyColNames);
```

5. Create an array of type `String` to hold the name of the columns to be updated. The following code fragment specifies the name of the `employee_id` column:

```
String[] updateColNames = new String[2];
updateColNames[0] = "SALARY";
updateColNames[1] = "JOB_ID";
```

6. Set the XML save object to the list of columns to be updated. The following statement performs this task:

```
sav.setUpdateColumnList(updateColNames);
```

7. Update the rows specified in the input XML document. The following code fragment creates a URL from the file name specified on the command line:

```
sav.updateXML(sav.getURL(argv[0]));
```

8. Close the XML save object:

```
sav.close();
```

21.3.7.2.1 Running the testUpdateList Program

The `testUpdateList` program is described.

Suppose that you use the sample XML document `upd_emp2.xml` to store new data for employees Steven King, who has an employee ID of 100, and William Gietz, who has an employee identifier (ID) of 206. You pass `upd_emp2.xml` as an argument to the `testUpdateList` program:

```
java testUpdateList upd_emp2.xml
```

In this example, the program generates two `UPDATE` statements. For the first `ROW` element, the program generates this statement:

```
UPDATE hr.employees SET salary = 8350 AND job_id = 'AC_ACCOUNT'
  WHERE employee_id = 100;
```

For the second `ROW` element the program generates this statement:

```
UPDATE hr.employees SET salary = 25000 AND job_id = 'AD_PRES'
  WHERE employee_id = 206;
```

21.3.8 Deleting Rows using XSU

When deleting from XML documents, you can specify a list of key columns. XSU uses these columns in the `WHERE` clause of the `DELETE` statement. If you do not supply the key column names, then XSU creates a new `DELETE` statement for each `ROW` element of the XML document.

The list of columns in the `WHERE` clause of the `DELETE` statement matches those in the `ROW` element.

Copyright © 1999, 2025, Oracle and/or its affiliates.

21.3.8.1 Deleting by Row with OracleXMLSave

The `testDeleteRow.java` demo program accepts an XML document file name as input and deletes the rows corresponding to the elements in the document.

The `testDeleteRow.java` program follows these steps:

1. Create a JDBC OCI connection. The program invokes the same `getConnection()` method used by the previous examples in this chapter:

   ```
   Connection conn = getConnection("hr","password");
   ```

2. Create an XML save object. You initialize the object by passing it the `Connection` reference and the name of the table on which you want to perform DML. The following example shows this technique:

   ```
   OracleXMLSave sav = new OracleXMLSave(conn, "employees");
   ```

3. Delete the rows specified in the input XML document. The following code fragment creates a URL from the file name specified on the command line:

   ```
   sav.deleteXML(sav.getURL(argv[0]));
   ```

4. Close the XML save object:

   ```
   sav.close();
   ```

21.3.8.1.1 Running the testDelete Program

The `testDelete` program is described.

This section shows how to delete the employees 7400 and 7369 that you added in Inserting Rows with OracleXMLSave.

To make this example work correctly, connect to the database and disable a constraint on the `hr.job_history` table:

```
CONNECT hr
ALTER TABLE job_history
  DISABLE CONSTRAINT JHIST_EMP_FK;
EXIT
```

Now pass `upd_emp.xml` to the `testDeleteRow` program:

```
java testDeleteRow upd_emp.xml
```

The program forms the `DELETE` statements based on the tag names present in each `ROW` element in the XML document. It executes these statements:

```
DELETE FROM hr.employees WHERE salary = 3250 AND employee_id = 7400;
DELETE FROM hr.employees WHERE job_id = 'SA_REP' AND MANAGER_ID = 145
  AND employee_id = 7369;
```

21.3.8.2 Deleting by Key with OracleXMLSave

To use only the key values as predicates on the `DELETE` statement, invoke the `OracleXMLSave.setKeyColumnList()` method. This approach limits the number of elements used to identify a row, which has the benefit of improving performance by caching the `DELETE` statement and batching transactions. The `testDeleteKey.java` program shows this technique.

The `testDeleteKey.java` program follows these steps:

Copyright © 1999, 2025, Oracle and/or its affiliates.

1. Create a JDBC OCI connection. The program invokes the same `getConnection()` method used by the previous examples in this chapter:

   ```
   Connection conn = getConnection("hr","password");
   ```

2. Create an XML save object. You initialize the object by passing it the `Connection` reference and the name of the table on which you want to perform DML. The following example shows this technique:

   ```
   OracleXMLSave sav = new OracleXMLSave(conn, "employees");
   ```

3. Create an array of type `String` to hold the name of the primary key column in the table. The array contains only one element. The following code fragment specifies the name of the `employee_id` column:

   ```
   String [] colNames = new String[1];
   colNames[0] = "EMPLOYEE_ID";
   ```

4. Set the XML save object to the primary key specified in the array. The following statement passes a reference to the `colNames` array to the `OracleXMLSave.setKeyColumnList()` method:

   ```
   sav.setKeyColumnList(keyColNames);
   ```

5. Delete the rows specified in the input XML document. The following code fragment creates a URL from the file name specified on the command line:

   ```
   sav.deleteXML(sav.getURL(argv[0]));
   ```

6. Close the XML save object:

   ```
   sav.close();
   ```

21.3.8.2.1 Running the testDeleteKey Program

The `testDeleteKey` program is described.

This section shows how to delete employees 7400 and 7369 that you added in Updating Key Columns Using OracleXMLSave. If you deleted these employees in the previous example, you can add them back to the `employees` table:

```
java testInsert new_emp.xml
java testInsert new_emp2.xml
```

Delete employees 7400 and 7369 by passing the same `upd_emp.xml` document to the `testDeleteRow` program:

```
java testDeleteKey upd_emp.xml
```

The program forms this single generated `DELETE` statement:

```
DELETE FROM hr.employees WHERE employee_id=?;
```

The program executes these `DELETE` statements, one for each employee:

```
DELETE FROM hr.employees WHERE employee_id = 7400;
DELETE FROM hr.employees WHERE employee_id = 7369;
```

21.3.9 Handling XSU Java Exceptions

XSU catches all exceptions that occur during processing and throws `oracle.xml.sql.OracleXMLSQLException`, which is a generic runtime exception. The invoking

Copyright © 1999, 2025, Oracle and/or its affiliates.

program need not catch this exception if it can still perform the appropriate action. The exception class provides methods to get error messages and any parent exceptions.

21.3.9.1 Getting the Parent Exception

The `testException.java` demo program throws a runtime exception and then gets the parent exception by invoking `Exception.getParentException()`.

Running the program generates this error message:

```
Caught SQL Exception:ORA-00904: "SD": invalid identifier
```

21.3.9.2 Raising a No Rows Exception

When there are no rows to process, XSU returns a `null` string. You can throw an exception each time there are no more rows, however, so that a program can process this exception using exception handlers.

When a program invokes `OracleXMLQuery.setRaiseNoRowsException()`, XSU raises an `oracle.xml.sql.OracleXMLSQLNoRowsException` whenever there are no rows to generate for the output. This is a runtime exception and need not be caught.

The `noRowsTest.java` demo program instantiates the `pageTest` class defined in `pageTest.java`. The condition to check the termination changed from checking whether the result is `null` to an exception handler.

The `noRowsTest.java` program creates a `pageTest` object and initializes it with a SQL query. The program proceeds as follows:

1. Configure the query object or raise a no rows exception. The following code fragment shows this technique:

   ```
   pageTest test = new pageTest("SELECT * from employees");
   ...
   test.qry.setRaiseNoRowsException(true);
   ```

2. Loop through the result set infinitely, retrieving ten rows at a time. When no rows are available, the program throws an exception. The following code fragment invokes `pageTest.nextPage()`, which scrolls through the result set ten rows at a time:

   ```
   try
   {
     while(true)
       System.out.println(test.nextPage());
   }
   ```

3. Catch the no rows exception and print "END OF OUTPUT". The following code shows this technique:

   ```
   catch(oracle.xml.sql.OracleXMLSQLNoRowsException e)
   {
     System.out.println(" END OF OUTPUT ");
     try
     {
       test.close();
     }
     catch ( Exception ae )
     {
       ae.printStackTrace(System.out);
     }
   }
   ```


Copyright © 1999, 2025, Oracle and/or its affiliates.

After compiling the program, run it from the command line:

```
java noRowsTest
```

21.4 Tips and Techniques for Programming with XSU

This section provides tips and techniques for writing programs with XSU.

21.4.1 How XSU Maps Between SQL and XML

The mapping between SQL and XML is described.

The fundamental component of a table is a column, whereas the fundamental components of an XML document are elements and attributes. How do tables map to XML documents? For example, if the `hr.employees` table has a column called `last_name`, how is this structure represented in XML: as an `<EMPLOYEES>` element with a `last_name` attribute or as a `<LAST_NAME>` element within a different root element? This section answers such questions by describing how SQL maps to XML and the reverse.

21.4.1.1 Default SQL-to-XML Mapping

The default mapping of SQL data to XML data is described.

To display data from some column of the `hr.employees` table as an XML document, run XSU at the command line:

```
java OracleXML getXML -user "hr/password" -withschema \
  "SELECT employee_id, last_name, hire_date FROM employees"
```

XSU outputs an XML document based on the input query. The root element of the document is `<DOCUMENT>`. The following shows sample output, with extraneous lines replaced by comments:

```
<?xml version = '1.0'?>
<DOCUMENT xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <!-- children of schema element ... -->
   </xsd:schema>
   <ROWSET xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="#/DOCUMENT/xsd:schema[not(@targetNamespace)]">
      <ROW num="1">
         <EMPLOYEE_ID>100</EMPLOYEE_ID>
         <LAST_NAME>King</LAST_NAME>
         <HIRE_DATE>6/17/1987 0:0:0</HIRE_DATE>
      </ROW>
      <!-- additional rows ... -->
   </ROWSET>
</DOCUMENT>
```

In the generated XML, the rows returned by the SQL query are children of the `<ROWSET>` element. The XML document has these features:

- The `<ROWSET>` element has zero or more `<ROW>` child elements corresponding to the number of rows returned. If the query generates no rows, then no `<ROW>` elements are included; if the query generates one row, then one `<ROW>` element is included, and so forth.

Copyright © 1999, 2025, Oracle and/or its affiliates.

- Each <ROW> element contains data from one table row. Specifically, each <ROW> element has one or more child elements whose names and content are identical to the database columns specified in the SELECT statement.

21.4.1.1.1 XML Mapping Against an Object-Relational Schema

XSU can generate an XML document from an object-relational schema.

Run the `createObjRelSchema.sql` script in SQL*Plus to set up and populate an object-relational schema. The schema contains a `dept1` table with two columns that employ user-defined types.

You can query the `dept1` table by invoking XSU from the command line:

```
% java OracleXML getXML -user "hr/password" -withschema "SELECT * FROM dept1"
```

XSU returns the XML document shown in Example 21-3, which is altered so that extraneous lines are replaced by comments.

As in the previous example, the mapping is canonical, that is, <ROWSET> contains <ROW> child elements, which in turn contain child elements corresponding to the columns in `dept1`. For example, the <DEPTNAME> element corresponds to the `dept1.deptname` column. The elements corresponding to scalar type columns contain the data from the columns.

Example 21-3 XSU-Generated Sample Document

```
<?xml version='1.0'?>
<DOCUMENT xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <schema targetNamespace="http://xmlns.oracle.com/xdb/SYSTEM"
           xmlns="http://www.w3.org/2001/XMLSchema"
           xmlns:SYSTEM="http://xmlns.oracle.com/xdb/SYSTEM">
   <!-- children of schema element ... -->
   </xsd:schema>
   <ROWSET xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="#/DOCUMENT/xsd:schema[not(@targetNamespace)]">
      <ROW num="1">
         <DEPTNO>120</DEPTNO>
         <DEPTNAME>Treasury</DEPTNAME>
         <DEPTADDR>
            <STREET>2004 Charade Rd</STREET>
            <CITY>Seattle</CITY>
            <STATE>WA</STATE>
            <ZIP>98199</ZIP>
         </DEPTADDR>
         <EMPLIST>
            <EMPLIST_ITEM>
               <EMPLOYEE_ID>1</EMPLOYEE_ID>
               <LAST_NAME>Mehta</LAST_NAME>
               <SALARY>6000</SALARY>
               <EMPLOYEE_ADDRESS>
                  <STREET>500 Main Road</STREET>
                  <CITY>Seattle</CITY>
                  <STATE>WA</STATE>
                  <ZIP>98199</ZIP>
               </EMPLOYEE_ADDRESS>
            </EMPLIST_ITEM>
         </EMPLIST>
      </ROW>
   </ROWSET>
</DOCUMENT>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

21.4.1.1.2 Default Mapping of Complex Type Columns to XML

The default mapping of complex-type columns to XML data is described.

The situation is more complex with elements corresponding to a complex-type column. In Example 21-3, <DEPTADDR> corresponds to the dept1.deptAddr column, which is of object type AddressType. Consequently, <DEPTADDR> contains child elements corresponding to the attributes specified in the type AddressType. The AddressType attribute street corresponds to the child XML element <STREET> and so forth. These subelements can contain data or subelements of their own, depending on whether the attribute they correspond to is of a simple or complex type.

21.4.1.1.3 Default Mapping of Collections to XML

The default mapping of database collections to XML data is described.

When dealing with elements corresponding to database collections, the situation is also different. In Example 21-3, the <EMPLIST> element corresponds to the emplist column of type EmployeeListType. Consequently, the <EMPLIST> element contains a list of <EMPLIST_ITEM> elements, each corresponding to an element of the collection. Note:

- The <ROW> elements contain a cardinality attribute num.
- If a particular column or attribute value is NULL, then for that row, the corresponding XML element is omitted.
- If a top-level scalar column name starts with the at sign (@) character, then the column is mapped to an XML attribute instead of an XML element.

21.4.1.2 Default XML-to-SQL Mapping

The default mapping of XML data to SQL data is described.

XML to SQL mapping is the reverse of SQL to XML mapping. Consider these differences when using XSU to map XML to SQL:

- When transforming XML to SQL, XSU ignores XML attributes. Thus, there is really no mapping of XML attributes to SQL.
- When transforming SQL to XML, XSU performs the mapping on a single ResultSet created by a SQL query. The query can span multiple database tables or views. When transforming XML into SQL, note:
  - 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, then you can use INSTEAD OF INSERT triggers.

If the XML document does not perfectly map to the target database schema, then you can perform these actions:

- Modify the target. Create an object-relational view over the target schema and make the view the new target.
- Modify the XML document by using XSLT to transform the XML document. You can register the XSLT stylesheet with XSU so that the incoming XML is automatically transformed before it attempts any mapping.

Copyright © 1999, 2025, Oracle and/or its affiliates.

- Modify XSU's XML-to-SQL mapping. You can instruct XSU to perform case-insensitive matching of XML elements to database columns or attributes. For example, you can instruct XSU to do this:
  - Use the name of the element corresponding to a database row instead of `ROW`.
  - Specify the date format to use when parsing dates in the XML document.

21.4.1.3 Customizing Generated XML

In some situations, you might need to generate XML with a specific structure. Because the desired structure might differ from the default structure of the generated XML document, you need to have some flexibility in this process.

21.4.1.3.1 Altering the Database Schema or SQL Query

You can perform source customizations by altering the SQL query or the database schema.

The simplest and most powerful source customizations include:

- In the database schema, create an object-relational view that maps to the desired XML document structure.
- In your query, do this:
  - Use cursor subqueries or cast-multiset constructs to create nesting in the XML document that comes from a flat schema.
  - Alias column and attribute names to get the desired XML element names.
  - Alias top-level scalar type columns with identifiers that begin with the at sign (@) to make them map to an XML attribute instead of an XML element. For example, executing these statement generates an XML document in which the `<ROW>` element has the attribute `empno`:

    ```
    SELECT employee_name AS "@empno",... FROM employees;
    ```

Consider the `customer.xml` document shown in Example 21-4.

Suppose you must design a set of database tables to store this data. Because the XML is nested more than one level, you can use an object-relational database schema that maps canonically to the preceding XML document. Run the `createObjRelSchema2.sql` script in SQL*Plus to create such a database schema.

You can load the data in the `customer.xml` document into the `customer_tab` table created by the script. Invoke XSU for Java from the command line:

```
java OracleXML putXML -user "hr/password" -fileName customer.xml customer_tab
```

To load `customer.xml` into a database schema that is not object-relational, you can create objects in views on top of a standard relational schema. For example, you can create a relational table that contains the necessary columns, then create a customer view that contains a customer object on top of it, as shown in the `createRelSchema.sql` script in Example 21-5.

You can load data into `customer_view`:

```
java OracleXML putXML -user "hr/password" -fileName customer.xml customer_view
```

Alternatively, you can flatten your XML with XSLT and then insert it directly into a relational schema. However, this is the least recommended option.

Copyright © 1999, 2025, Oracle and/or its affiliates.

To map a particular column or a group of columns to an XML attribute instead of an XML element, you can create an alias for the column name and prepend the at sign (@) before the name of this alias. For example, you can use the `mapColumnToAtt.sql` script to query the `hr.employees` table, rendering `employee_id` as an XML attribute.

You can run the `mapColumnToAtt.sql` script from the command line:

```
java OracleXML getXML -user "hr/password" -fileName "mapColumnToAtt.sql"
```

> **Note**
>
> All attributes must appear *before* any nonattribute.

Example 21-4 customer.xml

```
<?xml version = "1.0"?>
<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>
```

Example 21-5 createRelSchema.sql

```
CREATE TABLE hr.cust_tab
 ( customerid NUMBER(10),
   firstname VARCHAR2(20),
   lastname VARCHAR2(20),
   street VARCHAR2(40),
   city VARCHAR2(20),
   state VARCHAR2(20),
   zip VARCHAR2(20)
 );

CREATE VIEW customer_view
AS
SELECT customer_type(customerid, firstname, lastname,
       address_type(street,city,state,zip)) customer
FROM cust_tab;
```

21.4.1.3.2 Modifying XSU

XSU lets you modify the rules that it uses to transform SQL data into XML.

You can make any of these changes when mapping SQL to XML:

- Change or omit the `<ROWSET>` or `<ROW>` tag.
- Change or omit the attribute `num`, which is the cardinality attribute of the `<ROW>` element.

Copyright © 1999, 2025, Oracle and/or its affiliates.

- Specify the case for the generated XML element names.
- Specify that XML elements corresponding to elements of a collection must have a cardinality attribute.
- Specify the format for dates in the XML document.
- Specify that null values in the XML document must be indicated with a nullness attribute rather than by omitting the element.

21.4.2 How XSU Processes SQL Statements

How XSU processes SQL statements is described.

21.4.2.1 How XSU Queries the Database

XSU executes SQL queries and retrieves the `ResultSet` from the database. XSU then acquires and analyzes metadata about the `ResultSet`.

Using the mapping described in Default SQL-to-XML Mapping, XSU processes the SQL result set and converts it into an XML document.

XSU cannot handle certain types of queries, especially those that mix columns of type `LONG` or `LONG RAW` with `CURSOR()` expressions in the `SELECT` clause. `LONG` and `LONG RAW` are two examples of data types that JDBC accesses as streams and whose use is deprecated. If you migrate these columns to `CLOBs`, then the queries succeed.

21.4.2.2 How XSU Inserts Rows

The steps that XSU performs when inserting an XML document into a table or view are described.

When inserting the contents of an XML document into a table or view, XSU does the following:

1. Retrieves metadata about the target table or view.
2. Generates a SQL `INSERT` statement based on the metadata. For example, assume that the target table is `dept1` and the XML document is generated from `dept1`. XSU generates this `INSERT` statement:

   ```
   INSERT INTO dept1 (deptno, deptname, deptaddr, emplist) VALUES (?,?,?,?)
   ```

3. Parses the XML document, and for each record, it binds the appropriate values to the appropriate columns or attributes. For example, it binds the values for `INSERT` 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'),...)
   ```

4. Executes the statement. You can optimize `INSERT` processing to insert in batches and commit in batches.

Related Topics

- Default SQL-to-XML Mapping
  The default mapping of SQL data to XML data is described.

Copyright © 1999, 2025, Oracle and/or its affiliates.

> **See Also**
>
> Inserting Rows with OracleXMLSave for more detail on batching

21.4.2.3 How XSU Updates Rows

Updates and delete statements 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 if no triggers or constraints are on the table. With updates and deletes, the XML element can match more than one row if the matching columns are not key columns in the table.

For update statements, you must provide a list of key columns that XSU must identify the row to update. For example, assume that you have an XML document that contains this fragment:

```
<ROWSET>
  <ROW num="1">
    <DEPTNO>100</DEPTNO>
    <DEPTNAME>SportsDept</DEPTNAME>
  </ROW>
</ROWSET>
```

You want to change the `DEPTNAME` value from `Sports` to `SportsDept`. If you supply the `DEPTNO` as the key column, then XSU generates this `UPDATE` statement:

```
UPDATE dept1 SET deptname = ? WHERE deptno = ?
```

XSU binds the values in this way:

```
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.

Related Topics

- Updating Rows Using OracleXMLSave
  Examples show how to update the fields in a table or view. You supply the table or view name and an XML document. XSU parses the document (if a string is given) and creates one or more `UPDATE` statements into which it binds all of the values.

21.4.2.4 How XSU Deletes Rows

For row deletions, you can choose to provide a set of key columns, so that XSU can identify the rows to be deleted. If you do not provide a set of key columns then the `DELETE` statement tries to match all the columns in the document.

Assume that you pass this document to XSU:

```
<ROWSET>
 <ROW num="1">
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
  <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>
```

XSU builds a `DELETE` statement for each `ROW` element:

```
DELETE FROM dept1 WHERE deptno = ? AND deptname = ? AND deptaddr = ?
```

The binding is:

```
deptno   <- 100
deptname <- sports
deptaddr <- addresstype('100 redwood shores pkwy','redwood city','ca',
            '94065')
```

Related Topics

- Deleting Rows using XSU
  When deleting from XML documents, you can specify a list of key columns. XSU uses these columns in the `WHERE` clause of the `DELETE` statement. If you do not supply the key column names, then XSU creates a new `DELETE` statement for each `ROW` element of the XML document.

21.4.2.5 How XSU Commits After DML

By default, XSU performs no explicit commits. If `AUTOCOMMIT` is on, which is the default for a JDBC connection, then after each batch of statement executions XSU executes a `COMMIT`.

You can override this behavior by turning `AUTOCOMMIT` off and then using `setCommitBatch` to specify the number of statement executions before XSU commits. If an error occurs, then XSU rolls back to either the state the target table was in before the call to XSU, or the state after the last commit made during the current call to XSU.

Copyright © 1999, 2025, Oracle and/or its affiliates.

22

Using the TransX Utility

An explanation is given of how to use the TransX utility to transfer XML data to a database.

Related Topics

- Data Loading Format (DLF) Specification
  A description is given of version 1.0 of the Data Loading Format (DLF), which is the standard format for describing translated messages and seed data loaded into the database by the TransX utility.

22.1 Introduction to the TransX Utility

The TransX utility is described.

TransX Utility lets you transfer XML data to a database. TransX is an application of XML SQL Utility (XSU) that loads translated seed data and messages into a database schema.

TransX is particularly useful when handling multilingual Extensible Markup Language (XML). You can use TransX to add data to a database in multiple languages. The utility does this:

- Automatically manages the change variables, start sequences, and additional structured query language (SQL) statements that would otherwise require multiple inserts or sessions. Thus, translation vendors do not have to work with unfamiliar SQL and Procedural Language/Structured Query Language (PL/SQL) scripts.
- Automates character encoding. Consequently, loading errors due to incorrect encoding are impossible if the data file conforms with the XML standard.
- Reduces globalization costs by preparing strings to be translated, translating the strings, and loading them into the database.
- Minimizes translation data format errors and accurately loads the translation contents into predetermined locations in the database. When the data is in a predefined format, the TransX utility validates it.
- Eliminates syntax errors due to varying Globalization Support settings.
- Does not require the `UNISTR` construct for every piece of `NCHAR` data.

> **Note**
>
> TransX runs as the authenticated user. Care must be taken to review data files and to load data files only from a trusted source.

22.1.1 Prerequisites for Using the TransX Utility

Prerequisites for using the TransX utility are described.

This chapter assumes that you are familiar with XML SQL Utility (XSU) because TransX is an application of XSU.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Related Topics

- Using the XML SQL Utility
  An explanation is given of how to use the Extensible Markup Language (XML) SQL Utility (XSU).

22.1.2 TransX Utility Features

Topics here include simplified multilingual data loading, simplified data format support, and other TransX Utility features.

22.1.2.1 Simplified Multilingual Data Loading

The traditional translation data loading method is to change environment variable `NLS_LANG` when switching load files. This sets the language and territory used by client applications and the database server. It also sets the client character set, which is used for data entered or displayed by a client program.

When inserting multilingual data or data translations into a database, or when encoding, each XML file requires validation.

In the traditional method, each load file is encoded in a character set suitable for its language, which is necessary because translations must be performed in the same file format—typically in a SQL script—as the original. The `NLS_LANG` setting changes as files are loaded to adapt to the character set that corresponds to the language. As well as consuming time, this approach is error-prone because the encoding metadata is separate from the data itself.

With the TransX utility you use an XML document with a predefined format called a data set. The data set contains the encoding information and the data so that you can transfer multilingual data without changing `NLS_LANG` settings. The TransX utility frees development and translation groups by maintaining the correct character set while loading XML data into the database.

> **See Also**
>
> *Oracle AI Database Globalization Support Guide* to learn about the `NLS_LANG` environment variable

22.1.2.2 Simplified Data Format Support and Interface

The TransX Utility provides a command-line interface and a programmable application programming interface (API). The utility complies with a data format that is the canonical method for the representation of seed data loaded into the database. The format is easy to understand and simplified for use by translation groups.

The format specification defines how translators can describe the data so that it is loaded in an expected way. You can represent the values in the data set with scalar values or expressions such as constants, sequences, and queries.

22.1.2.3 Additional TransX Utility Features

Other useful TransX Utility features are described.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 22-1 TransX Utility Features

Feature	TransX Utility . . .
Command-line interface	Provides easy-to-use commands.
User API	Exposes a Java API.
Validation	Validates the data format and reports errors.
White space handling	Does not consider white space characters in the data set as significant unless otherwise specified in various granularity.
Unloading	Exports the result into the standard data format based on an input query.
Intimacy with translation exchange format	Enables transformation to and from translation exchange format.
Localized user interface	Provides messages in many languages.

22.2 Using the TransX Utility: Overview

Topics here describe how to use the TransX utility.

22.2.1 Using the TransX Utility: Basic Process

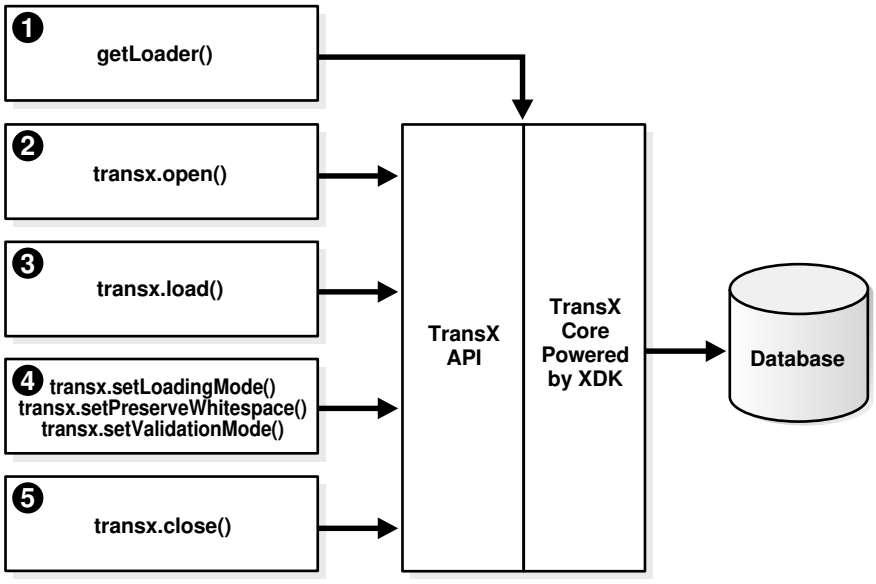
The TransX API basic process is described.

TransX is accessible through this API:

- `oracle.xml.transx.loader` class, which contains the `getLoader()` method to get a TransX instance
- `oracle.xml.transx.TransX` interface, which is the TransX API

Figure 22-1 shows the basic process for using the TransX API to transfer XML to a database.

Figure 22-1 Basic Process of a TransX Application



Copyright © 1999, 2025, Oracle and/or its affiliates.

The basic process of a TransX application is:

1. Create a TransX loader object. Instantiate the `TransX` class by invoking `getLoader()`:

   ```
   TransX  transx = loader.getLoader();
   ```

2. Start a data loading session by supplying database connection information using `TransX.open()`. You create a session by supplying the Java Database Connectivity (JDBC) connect string, database user name, and database password. You can create the connection in one of these ways:

   - Using the JDBC Oracle Call Interface (OCI) driver. The following code fragment shows this technique and connects with the supplied user name and password:

     ```
     transx.open( "jdbc:oracle:oci8:@", user, passwd );
     ```

   - Using the JDBC thin driver. The thin driver is written in pure Java and can be called from any Java program. The following code fragment shows this technique and connects:

     ```
     transx.open( "jdbc:oracle:thin:@//myhost:1521/myservicename", user,passwd);
     ```

     The thin driver requires the host name (`myhost`), port number (`1521`), and the service name (`myservicename`). The database must have an active Transmission Control Protocol/Internet Protocol (TCP/IP) listener.

   > **Note**
   >
   > If you are validating only your data format, you do not have to establish a database connection because the validation is performed by TransX. Thus, you can invoke the `TransX.validate()` method without a preceding `open()` invocation.

3. Configure the TransX loader. Table 22-2 describes configuration methods.

Table 22-2 TransX Configuration Methods

Method	Description
`setLoadingMode()`	Sets the operation mode on duplicates. The mode determines TransX behavior when there are one or more existing rows in the database whose values in the key columns are the same as those in the data set to be loaded. You can specify the constants `EXCEPTION_ON_DUPLICATES`, `SKIP_DUPLICATES`, or `UPDATE_DUPLICATES` in class `oracle.xml.transx.LoadingMode`. By default the loader skips duplicates.
`setNormalizeLangTag()`	Sets the case of language tag. By default the loader uses the style specified in the `normalize-langtag` attribute on Data Loading Format (DLF).
`setPreserveWhitespace()`	Specifies how the loader handles white space. The default is `FALSE`, which means that the loader ignores the type of white space characters in the data set and loads them as space characters. The loader treats consecutive white space characters in the data set as one space character.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 22-2 (Cont.) TransX Configuration Methods

Method	Description
`setValidationMode()`	Sets the validation mode. The default is TRUE, which means that the loader performs validation of the data set format against the canonical schema definition on each `load()` invocation. The validation mode is disabled only if the data set has already been validated.

The following example specifies that the loader must skip duplicate rows and not validate the data set:

```
transx.setLoadingMode( LoadingMode.SKIP_DUPLICATES );
transx.setValidationMode( false );
```

4. Load the data sets by invoking `TransX.load()`. The same JDBC connection is used during the iteration of the load operations. For example, load three data sets:

```
String  datasrc[] = {"data1.xml", "data2.xml", "data3.xml"};
...
for ( int i = 0 ; i < datasrc.length ; i++ )
{
  transx.load( datasrc[i] );
}
```

5. Close the loading session by invoking `TransX.close()`. This method invocation closes the database connection:

```
transx.close();
```

> **See Also**
>
> - *Oracle AI Database Java Developer's Guide* to learn about Oracle JDBC
> - *Oracle AI Database XML Java API Reference* to learn about `TransX` classes and methods

22.2.2 Running the TransX Utility Demo Programs

Demo programs for the TransX utility are included in `$ORACLE_HOME/xdk/demo/java/transx`.

Table 22-3 describes the XML files and programs that you can use to test the utility.

Table 22-3 TransX Utility Sample Files

File	Description
`README`	A text file that describes how to set up the TransX demos.
`emp-dlf.xml`	A sample output file. The following command generates a file `emp.xml` that contains all data in the table emp: `transx -s "jdbc:oracle:thin:@//myhost:1521/myservicename" user -pw emp.xml emp` The `emp-dlf.xml` file must be identical to `emp.xml`.
`txclean.sql`	A SQL file that drops the tables and sequences created for the demo.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 22-3 (Cont.) TransX Utility Sample Files

File	Description
`txdemo1.java`	A sample Java application that creates a JDBC connection and loads three data sets into the database.
`txdemo1.sql`	A SQL script that creates two tables and a sequence for use by the demo application.
`txdemo1.xml`	A sample data set.

Documentation for how to compile and run the sample programs is located in the `README`. The basic steps are:

1. Change into the `$ORACLE_HOME/xdk/demo/java/transx` directory (UNIX) or `%ORACLE_HOME%\xdk\demo\java\transx` directory (Windows).
2. Make sure that your environment variables are set as described in Setting Up the XDK for Java Environment. Oracle recommends that you set the `$ORACLE_SID` (UNIX) or `%ORACLE_SID%` (Windows) environment variables to the default database.

   > **Note**
   >
   > For security, do not expose passwords in command-line interfaces. If "`-pw`" is specified instead of the password in `TransX`, the user is prompted for the password [`Enter password :`]. When the user types the password, it is not echoed; instead, "*"s is printed in the command window.

3. Set up the sample database objects by executing `txdemo1.sql`. Connect to the database and run the `txdemo1.sql` script:

   ```
   @txdemo1
   ```

4. Run the TransX utility from the command line. This example shows how to connect with the Java thin driver, where your host is `myhost`, your port is `1521`, and your service name is `myservicename`. Enter the user name where the token *`user`* appears. You can execute this command to load data set `txdemo1.xml`:

   ```
   transx "jdbc:oracle:thin:@//myhost:1521/myservicename" user -pw txdemo1.xml
   ```

   When the operation is successful, nothing is printed out on your terminal.

5. Query the database to determine whether the load was successful. For example:

   ```
   SELECT * FROM i18n_messages;
   ```

6. Drop the demo objects to prepare for another test. Connect to the database and run the `txclean.sql` script:

   ```
   @txclean
   ```

7. Compile the Java demo program. For example:

   ```
   javac txdemo1.java
   ```

8. Run the Java program, using the same JDBC and database connection data that you used when invoking the command-line interface. For example:

   ```
   java txdemo1 "jdbc:oracle:thin:@//myhost:1521/myservicename" user -pw\
                txdemo1.xml
   ```


Copyright © 1999, 2025, Oracle and/or its affiliates.

Perform the same query test (Step 5) and cleanup operation (Step 6) as before.

9. Run the TransX Utility to unload data into the predefined XML format. For example:

```
transx -s "jdbc:oracle:thin:@//myhost:1521/myservicename" user -pw emp.xml emp
```

Compare the data in `emp.xml` with `emp-dlf.xml`.

> **Note**
>
> For simplicity in demonstrating this feature, this example does not perform the password management techniques that a deployed system normally uses. In a production environment, follow the Oracle Database password management guidelines, and disable any sample accounts. See *Oracle AI Database Security Guide* for password management guidelines and other security recommendations.

22.2.3 Using the TransX Command-Line Utility

TransX utility is packaged with Oracle Database. By default, the Oracle Universal Installer installs the utility on disk.

As explained in XDK for Java Component Dependencies, the TransX library is `$ORACLE_HOME/lib/xml.jar` (UNIX) and `%ORACLE_HOME%\lib\xml.jar` (Windows).

You can run the TransX utility from the operating system command line with this syntax:

```
java oracle.xml.transx.loader
```

Oracle XML Developer's Kit (XDK) includes a script version of TransX named `$ORACLE_HOME/bin/transx` (UNIX) and `%ORACLE_HOME%\bin\transx.bat` (Windows). Assuming that your `PATH` variable is set correctly, you can run TransX:

```
transx options parameters
transx.bat options parameters
```

For example, this command shows valid syntax:

```
transx -s "jdbc:oracle:thin:@//myhost:1521/myservicename" user -pw emp.xml emp
```

22.2.3.1 TransX Utility Command-Line Options

The command-line options for the TransX Utility are described.

Table 22-4 TransX Utility Command-Line Options

Option	Meaning	Description
`-u`	Update existing rows.	Does not skip existing rows but updates them. To exclude a column from the update operation, set the `useforupdate` attribute to `no`.
`-e`	Raise exception if a given row already exists in the database.	Raises an exception if a duplicate row is found. By default, TransX skips duplicate rows. Rows are considered duplicate if the values for lookup-key column(s) in the database and the data set are the same.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 22-4 (Cont.) TransX Utility Command-Line Options

Option	Meaning	Description
`-x`	Print database data in the predefined format.	Similar to the `-s` option, it causes the utility to perform the opposite operation of loading. Unlike the `-s` option, it prints to `stdout`. Redirecting this output to a file is discouraged because intervention of the operating system may cause data loss due to unexpected transcoding.
`-s`	Save database data to a file in the predefined format.	Performs unloading. TransX Utility queries the database, formats the result into the predefined XML format, and stores it under the specified file name.
`-p`	Print the XML to load.	Prints out the data set for insert in the canonical format of XSU.
`-t`	Print the XML for update.	Prints out the data set for update in the canonical format of XSU.
`-o`	Omit validation (as the data set is parsed it is validated by default).	Causes TransX Utility to skip the format validation, which is performed by default.
`-v`	Validate the data format and exit without loading.	Causes TransX Utility to perform validation and exit.
`-w`	Preserve white space.	Causes TransX Utility to treat white space characters (such as \t, \r, \n, and ' ') as significant. The utility condenses consecutive white space characters in string data elements into one space character by default.
`-l`	Set the case of language tag.	Causes TransX Utility to override the style of normalizing the case of language tag specified in the `normalize-langtag` attribute on DLF or the `setNormalizeLangTag()` method on the TransX API. Valid options are `-ls`, `-lu` and `-ll` for standard, uppercase and lowercase, respectively.

Command-line option exceptions:

- `-u` and `-e` are mutually exclusive.
- `-v` must be the only option followed by data, as shown in the examples.
- `-x` must be the only option followed by connect information and a SQL query, as shown in the examples.

Omitting all arguments produces the display of the usage information shown in Table 22-4.

22.2.3.2 TransX Utility Command-Line Parameters

The command-line parameters for the TransX utility are described.

Table 22-5 TransX Utility Command-Line Parameters

Parameter	Description
`connect_string`	The JDBC connect string. See *Oracle AI Database JDBC Developer's Guide*,

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 22-5 (Cont.) TransX Utility Command-Line Parameters

Parameter	Description
`username`	Database user name (schema).
`password`	Password for the database user, or "`-pw`".
`datasource`	An XML document specified by file name or URL.
`options`	Described in Table 22-4.

> **See Also**
>
> *Oracle AI Database XML Java API Reference* for complete details of the `TransX` interface

22.3 Loading Data with the TransX Utility

You can use the TransX utility to populate a database with multilingual data. To transfer data in and out of a database schema, you create a data set that maps to this schema. A scenario is described in which you use TransX to organize translated application messages in a database.

22.3.1 Storing Messages in the Database

Data that is specific to a particular region and shares a common language and cultural conventions must be organized with a resource management facility that can retrieve locale-specific information. A database is often used to store such data because of easy maintenance and flexibility.

To build an internationalized system, it is essential to decouple localizable resources from business logic. A typical example of such a resource is translated text information.

Assume that you create the table with the structure and content shown in Example 22-1 and insert data.

The column `language_id` is defined in this table so that applications can retrieve messages based on the preferred language of the end user. It contains abbreviations of language names to identify the language of messages.

Example 22-2 shows sample data for the table.

> **See Also**
>
> *Oracle AI Database Globalization Support Guide* for Oracle language abbreviations

Copyright © 1999, 2025, Oracle and/or its affiliates.

Example 22-1 Structure of Table translated_messages

```
CREATE TABLE translated_messages
(
  MESSAGE_ID       NUMBER(4)
     CONSTRAINT tm_mess_id_nn NOT NULL
, LANGUAGE_ID      VARCHAR2(42)
, MESSAGE          VARCHAR2(200)
);
```

Example 22-2 Query of translated_messages

```
MESSAGE_ID  LANGUAGE_ID  MESSAGE
----------  -----------  --------------------------------
1           us           Welcome to System X
2           us           Please enter username and password
```

22.3.2 Creation of a Data Set in a Predefined Format

An example shows an XML document that represents the `translated_messages` table.

Data Loading Format (DLF) Specification describes the complete syntax of the DLF language. This language is used to create a DLF document that provides the input to TransX.

Given the data set (the input data) in the canonical format, the TransX Utility loads the data into the designated locations in the database. TransX does not create the database objects: you must create the tables or views before attempting to load data.

An XML document that represents the `translated_messages` table created in Example 22-1 looks something like Example 22-3. The data set reflects the structure of the target table, which in this case is called `translated_messages`.

Example 22-3 example.xml

```
<?xml version="1.0"?>
<table name="translated_messages">
  <!-- Specify the unique identifier -->
  <lookup-key>
    <column name="message_id" />
    <column name="language_id" />
  </lookup-key>
  <!-- Specify the columns into which data will be inserted -->
  <columns>
    <column name="message_id"  type="number"/>
    <column name="language_id" type="string" constant="us" translate="yes"/>
    <column name="message"     type="string" translate="yes"/>
  </columns>
  <!-- Specify the data to be inserted -->
  <dataset>
    <row>
      <col name="message_id">1</col>
      <col name="message" translation-note="dnt'X'">Welcome to System X</col>
    </row>
    <row>
      <col name="message_id">2</col>
      <col name="message">Please enter username and password</col>
    </row>
    <!-- ... -->
  </dataset>
</table>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

22.3.2.1 Format of the Input XML Document

The format of the input XML document is described.

The XML document in Example 22-3 starts with this declaration:

```
<?xml version="1.0"?>
```

Its root element <table>, which has an attribute that specifies the name of the table, encloses all the other elements:

```
<table name="translated_messages">
...
</table>
```

As explained in Elements in DLF, the <table> element contains three subsections:

- Lookup Key Elements
- Metadata Elements
- Data Elements

The preceding sections map to elements in Example 22-3:

```
<lookup-key>...</lookup-key>
<columns>...</columns>
<dataset>...</dataset>
```

The lookup keys are columns used to evaluate rows if they already exist in the database. Because you want a pair of message and language identifiers to identify a unique string, the document lists the corresponding columns. Thus, the message_id, language_id, and message columns in table translated_messages map to the attributes in the <column> element:

```
<column name="message_id"  type="number"/>
<column name="language_id" type="string" constant="us" translate="yes"/>
<column name="message"     type="string" translate="yes"/>
```

The columns section must mirror the table structure because it specifies which piece of data in the data set section maps to which table column. The column names must be consistent throughout the XML data set and database. You can use the <column> attributes in Table 22-6 to describe the data to be loaded. These attributes form a subset of the DLF attributes described in Attributes in DLF.

Table 22-6 <column> Attributes

Attribute	Description	Example
type	Specifies the data type of a column in the data set. This attribute specifies the kind of text contained in the <col> element in the data set. Depending on this type, the data loading tool applies different data type conventions to the data.	`<column name="col" type="string" />`
constant	Specifies a constant value. A column with a fixed value for each row does not have to repeat that same value.	`<column name="col" type="string" constant="us" />`

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 22-6 (Cont.) <column> Attributes

Attribute	Description	Example
language	The language attribute indicates that the column is the language column, which stores a language tag. It works in the same way as the constant attribute, except for the role to declare the column is the language column.	`<column name="language" type="string" language="us" />`
sequence	Specifies a sequence in the database used to fill in the value for this column.	`<column name="id" type="number" sequence="id_sq" />`
translate	Indicates whether the text of this column or parameter is to be translated.	`<column name="msg" type="string" translate="yes"/>`

The `constant` attribute of a `<column>` element specifies a value to be stored into the corresponding column for every row in the `<dataset>` section. Because this example is working in the original language, the `language_id` column is set to the value `us`.

Defining the Language Column

Alternatively, the `language_id` column may use the `language` attribute instead of the `constant` attribute. A DLF document with the `language` attribute can use the `lang` attribute in the xml namespace. A language column can use the "%x" placeholder to get its value from the standard `xml:lang` attribute at the root table element.Thus `translate="yes"` is not required, because the value "%x" does not have to be translated. The result of loading this DLF is the same as Example 10-3.

As explained in Table 23-10, the valid values for the `type` attribute are `string`, `number`, `date`, and `dateTime`. These values correspond to the data types defined in the XML schema standard, so each piece of data must conform to the respective data type definition. In particular, it is important to use the International Organization for Standardization (ISO) 8601 format for the `date` and `dateTime` data types, as shown in Table 22-7.

Table 22-7 date and dateTime Formats

Data Type	Format	Example
date	CCYY-MM-DD	2009-05-20
dateTime	CCYY-MM-DDThh:mm:ss	2009-05-20T16:01:37

Example 22-5 shows how you can represent a table row with `dateTime` data in a TransX data set.

Example 22-4 example.xml with a Language Attribute

```
<?xml version="1.0"?>
<table xml:lang="us" name="translated_messages">
  <!-- Specify the unique identifier -->
  <lookup-key>
    <column name="message_id" />
    <column name="language_id" />
  </lookup-key>
  <!-- Specify the columns into which data will be inserted -->
  <columns>
    <column name="message_id"  type="number"/>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
    <column name="language_id" type="string" language="%x"/>
    <column name="message"     type="string" translate="yes"/>
  </columns>
  <!-- Specify the data to be inserted -->
  <dataset>
    <row>
      <col name="message_id">1</col>
      <col name="message" translation-note="dnt'X'">Welcome to System X</col>
    </row>
    <row>
      <col name="message_id">2</col>
      <col name="message">Please enter username and password</col>
    </row>
    <!-- ... -->
  </dataset>
</table>
```

Example 22-5 dateTime Row

```
<row>
  <col name="article_id">12345678</col>
  <col name="author_id">10500</col>
  <col name="submission">2002-03-09T16:01:37</col>
  <col name="title">...</col>
  <!-- some columns follows -->
</row>
```

22.3.2.2 Specifying Translations in a Data Set

You can use the `translation` attribute to specify whether a column contains translated data.

This is explained in Attributes in DLF. In Example 22-3, two `<column>` elements use the `translate` attribute differently. The attribute for the `language_id` column specifies that the value of the `constant` attribute must be translated:

```
<column name="language_id" type="string" constant="us" translate="yes"/>
```

In contrast, this `translate` attribute requests translation of the data in the `<dataset>` section with a name that matches this column:

```
<column name="message"     type="string" translate="yes"/>
```

For example, the preceding element specifies that thesethis messages in the `<dataset>` section must be translated:

```
<col name="message" translation-note="dnt'X'">Welcome to System X</col>
<col name="message">Please enter username and password</col>
```

When translating messages for applications, you might decide to leave specified words or phrases untranslated. The `translation-note` attribute shown in the preceding example achieves this goal.

An Extensible Stylesheet Language Transformation (XSLT) processor can convert the preceding format into another format for exchanging translation data among localization service providers for use with XML-based translation tools. This transformation insulates developers from tasks such as keeping track of revisions, categorizing translatable strings into units, and so on.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Example 22-6 shows what (the beginning of) the document in Example 22-3 looks like after translation.

Example 22-7 shows what the document in Example 22-4 looks like after translation. Unlike the previous example, the column definition is not changed.

If you use a text editor or a traditional text-based translation tool during the translation process, it is important to maintain the encoding of the document. After a document is translated, it may be in a different encoding from the original. As explained in XML Declaration in DLF, If the translated document is in an encoding other than Unicode, then add the encoding declaration to the XML declaration on the first line. A declaration for non-Unicode encoding looks like these:

```
<?xml version="1.0" encoding="ISO-8859-15"?>
```

To ensure that the translation process does not lose syntactic integrity, process the document as XML. Otherwise, you can check the format by specifying the `-v` option of the command-line interface. If a syntactic error exists, the utility prints the location and description of the error. You must fix errors for the data transfer to succeed.

Example 22-6 example_es.xml

```
<?xml version="1.0"?>
<table xml:lang="es" name="translated_messages">
<!-- Specify the unique identifier -->
<lookup-key>
<column name="message_id" />
<column name="language_id" />
</lookup-key>
<!-- Specify the columns into which data will be inserted -->
<columns>
<column name="message_id" type="number"/>
<column name="language_id" type="string" constant="es"
translate="yes"/>
```

Example 22-7 example_es.xml with a Language Attribute

```
<?xml version="1.0"?>
<table xml:lang="es"  name="translated_messages">
  <!-- Specify the unique identifier -->
  <lookup-key>
    <column name="message_id" />
    <column name="language_id" />
  </lookup-key>
  <!-- Specify the columns into which data will be inserted -->
  <columns>
    <column name="message_id"  type="number"/>
    <column name="language_id" type="string" language="%x"/>
...
```

Related Topics

- Data Loading Format (DLF) Specification
  A description is given of version 1.0 of the Data Loading Format (DLF), which is the standard format for describing translated messages and seed data loaded into the database by the TransX utility.

Copyright © 1999, 2025, Oracle and/or its affiliates.

22.3.3 Loading the Data

The use of demo program `txdemo1.java` is described.

You can load the sample documents in Example 22-3 and Example 22-8 into the `translated_messages` table that you created in Example 22-1. Then, you can use the sample program in Example 22-8, which you can find in the TransX demo directory, to load the data.

The `txdemo1.java` program follows these steps:

1. Create a TransX loader object. For example:

   ```
   TransX  transx = loader.getLoader();
   ```

2. Open a data loading session. The first three command-line parameters are the JDBC connect string, database user name, and database password. These parameters are passed to the `TransX.open()` method. The program includes this statement:

   ```
   transx.open( args[0], args[1], args[2] );
   ```

3. Configure the TransX loader. The program configures the loader to skip duplicate rows and to validate the input data set. The program includes these statements:

   ```
   transx.setLoadingMode( LoadingMode.SKIP_DUPLICATES );
   transx.setValidationMode( false );
   ```

4. Load the data. The first three command-line parameters specify connection information; any additional parameters specify input XML documents. The program invokes the `load()` method for every specified document:

   ```
   for ( int i = 3 ; i < args.length ; i++ )
   {
     transx.load( args[i] );
   }
   ```

5. Close the data loading session. The program includes this statement:

   ```
   transx.close();
   ```

After compiling the program with `javac`, you can run it from the command line. The following example uses the Java thin driver to connect to instance `mydb` on port `1521` of computer `myhost`. It connects to the *user* schema and loads the XML documents `example.xml` and `example_es.xml`:

```
java txdemo1 "jdbc:oracle:thin:@//myhost:1521/mydb" user -pw example.xml
     example_es.xml
```

In building a multilingual software system, translations usually become available at a later stage of development. They also tend to evolve over time. To add messages to the database later, run the TransX utility again to add new rows in your data set definition. TransX recognizes which rows are new and inserts only the new messages based on the columns specified in the `<lookup-key>` section. If some messages are updated, then run TransX with the `-u` option to update existing rows with the data specified in XML, as shown in this example:

```
transx -u "jdbc:oracle:thin:@//myhost:1521/mydb" user -pw example.xml
       example_es.xml
```

Example 22-8 txdemo1.java

```
// Copyright (c) 2001 All rights reserved Oracle Corporation

import oracle.xml.transx.*;
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
public class txdemo1 {

  /**
   * Constructor
   */
  public txdemo1() {
  }

  /**
   * main
   * @param args
   *
   * args[0] : connect string
   * args[1] : username
   * args[2] : password
   * args[3+] : xml file names
   */
  public static void main(String[] args) throws Exception {

    // instantiate a transx class
    TransX  transx = loader.getLoader();

    // start a data loading session
    transx.open( args[0], args[1], args[2] );

    // specify operation modes
    transx.setLoadingMode( LoadingMode.SKIP_DUPLICATES );
    transx.setValidationMode( false );

    // load the dataset(s)
    for ( int i = 3 ; i < args.length ; i++ )
    {
      transx.load( args[i] );
    }

    // cleanup
    transx.close();
  }
}
```

22.3.4 Querying the Data

The result of querying table `translated_messages` is shown.

After using the program in Example 22-8 to load the data, you can query table `translated_messages` to see the result, which looks like this:

```
MESSAGE_ID  LANGUAGE_ID  MESSAGE
----------  -----------  ---------------------------------
1           us           Welcome to System X
1           es           Bienvenido al Sistema X
2           us           Please enter username and password
2           es           Porfavor entre su nombre de usuario y su contraseña
```

An application can retrieve a message in a specific language by using the `language_id` and `message_id` columns in a `WHERE` clause. For example, you can execute this query:

```
SELECT message
FROM   translated_messages
WHERE  message_id = 2
AND    language_id = 'es';
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
MESSAGE
---------------------------------
Porfavor entre su nombre de usuario y su contraseña
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

23
Data Loading Format (DLF) Specification

A description is given of version 1.0 of the Data Loading Format (DLF), which is the standard format for describing translated messages and seed data loaded into the database by the TransX utility.

Related Topics

- Using the TransX Utility
  An explanation is given of how to use the TransX utility to transfer XML data to a database.

23.1 Introduction to DLF

DLF defines a standard format for loading data with the TransX utility. It is intended to supersede loading data with SQL scripts. DLF provides these advantages:

- Format validation. Validation reduces errors during the translation and loading processes.
- Ease of use. The user does not have to maintain the character encoding of each data file to correspond with the language used in the data file.

DLF is based on the XML 1.0 specification.

> **Note**
>
> TransX runs as the authenticated user. Be sure to review your data files, and load data files only from a trusted source.

23.1.1 Naming Conventions for DLF

This section describes the naming conventions used in this document.

23.1.1.1 Elements and Attributes

Naming conventions for elements and attributes that are used in this document are described.

- Standard English letters
- Lowercase letters only
- Hyphen (-) may be used for concatenation
- Attribute names must be consistently defined throughout
- Industry-standard terminology must be followed wherever possible

23.1.1.2 Values

Values are case-sensitive except for some attribute values used for column names. All predefined attribute values are lowercase. No element values are defined by this specification.

Copyright © 1999, 2025, Oracle and/or its affiliates.

23.1.1.3 File Extensions

No file extension is recommended by this specification. A future version of this specification may recommend that documents use an extension that conforms with an 8.3 standard.

23.2 General Structure of DLF

Data Loading Format is XML, so it begins with an XML declaration. After the XML declaration comes the DLF document itself, enclosed within element `<table>`.

A DLF document is composed of these required sections:

- The `<lookup-key>` element contains a list of column names that determine whether existing rows in the database are duplicates of the rows in the data set definition included in the `<dataset>` element.
- The `<columns>` element contains metadata about the `<dataset>` element such as the names, data types, and attributes of columns.
- The `<dataset>` element contains a `<row>` element for each row, which in turn contains a `<col>` element that corresponds to a piece of data that is loaded in a database column. In this way a DLF document looks similar to the familiar tabular format in printing data in the database and allows easy editing.

DLF provides one optional section, which is enclosed within a `<translation>` element. This section may precede the required sections.

In addition, DLF provides information about TransX utility processing. Such information includes but is not limited to this:

- The `<query>` element is used to retrieve the value to be loaded to the column from a SQL query.
- The `sequence` attribute is used to retrieve the value to be loaded to the column from a sequence object in the database.
- The `constant` attribute is used to specify a constant value to the column.
- The `language` attribute is used to specify the language identifier to be loaded to the column.

23.2.1 Tree Structure of DLF

The possible structure of a DLF document is shown as a tree. Each element is represented as `<element_name>`, where *`element_name`* is the name of an element. Attributes have no markup. Each element and attribute is followed by notation indicating its possible occurrence.

Table 23-1 describes the occurrence notation.

Table 23-1 Notation for Occurrence of Attributes and Elements

Symbol	Meaning
1	one
+	one or more
?	zero or one
*	zero or more

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 23-1 (Cont.) Notation for Occurrence of Attributes and Elements

Symbol	Meaning
(a|b|c)	exactly one of a, b, and c

Example 23-1 shows the tree structure of a DLF document. The elements are described in Elements in DLF. The attributes are described in Attributes in DLF.

Example 23-1 DLF Tree Structure

```
<table>1
 |
 +---- lang?
 |
 +---- space?
 |
 +---- normalize-langtag?
 |
 +---- <translation>?
 |     |
 |     +---- <target>+
 |     |     |
 |     |     +---- <language ID>
 |     |
 |     +---- <restype>+
 |           |
 |           +---- name1
 |           |
 |           +---- expansion?
 |
 +---- <lookup-key>1
 |     |
 |     +---- <column>*
 |           |
 |           +---- name1
 |
 +---- <columns>1
 |     |
 |     +---- <column>+
 |           |
 |           +---- name1
 |           |
 |           +---- type1
 |           |
 |           +---- translate?
 |           |
 |           +---- translation-note?
 |           |
 |           +---- constant?
 |           |
 |           +---- language?
 |           |
 |           +---- sequence?
 |           |
 |           +---- virtual?
 |           |
 |           +---- useforupdate?
 |           |
 |           +---- maxsize?
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
|           |
|           +---- size-unit?
|           |
|           +---- restype?
|           |
|           +---- space?
|           |
|           +---- (<query>|<sql>)?
|                  |
|                  +---- text1
|                  |
|                  +---- <parameter>*
|                         |
|                         +---- id1
|                         |
|                         +---- (col|constant)1
|                         |
|                         +---- translate?
|                         |
|                         +---- trans-key?
|                         |
|                         +---- translation-note?
|
|
+---- <dataset>1
       |
       +---- <row>+
              |
              +---- space?
              |
              +---- <col>*
                     |
                     +---- space?
                     |
                     +---- name1
                     |
                     +---- trans-key?
                     |
                     +---- translation-note?
                     |
                     +---- <the text element for the data>
```

23.3 DLF Specifications

Topics here include XML declarations, entity references, elements, and attributes in DLF.

23.3.1 XML Declaration in DLF

The Extensible Markup Language (XML) declaration starts an XML entity. It indicates the XML version.

It can also declare the encoding of the file, as in this example:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
```

As in all XML files, the default encoding for a DLF file is assumed to be either 8-bit encoding of Unicode (UTF-8), which is a superset of the 7-bit ASCII character set, or 16-bit encoding of Unicode (UTF-16), which is conceptually 2-byte Universal Character Set (UCS-2) with

Copyright © 1999, 2025, Oracle and/or its affiliates.

surrogate pairs for code points above 65,535. Thus, for these character sets, the encoding declaration is not necessary. Furthermore, all XML parsers support these character sets. If the encoding is UTF-16, then the first character of the file must be the Unicode Byte-Order-Mark, #xFEFF, which indicates the endianness of the file.

Other character sets supported by Oracle XML parsers include all Oracle character sets and commonly used Internet Assigned Numbers Authority (IANA) character set and Java encodings. The names of these character sets can be found in the parser documentation. You must declare these with encoding declarations if the document does not have an external source of encoding information such as from the execution environment or the network protocol. Therefore, Oracle recommends that you use a Unicode character encoding so that you can dispense with the encoding declaration. The recommended practice is to encode the document in UTF-8 and use this declaration:

```
<?xml version="1.0" ?>
```

23.3.2 Entity References in DLF

XML predefines five entity references: `<`, `>`, `&`, `'`, and `"`.You must use entity references `<` and `&` in place of the characters they reference.

Table 23-2 Entity References

Entity Reference	Meaning
`<`	Less than sign (<)
`>`	Greater than sign (>)
`&`	Ampersand (&)
`'`	Apostrophe or single quotation mark (')
`"`	Straight, double quotation mark (")

23.3.3 Elements in DLF

Categories of DLF elements are described.

The DLF elements shown in Example 23-1 are divided into the categories described in Table 23-3. Attributes are shared among them. The attributes are described in Attributes in DLF.

Table 23-3 DLF Elements

Type of Element	Tag
Top-Level Table Element	`<table>`
Translation Elements	`<target>`, `<restype>`
Lookup Key Elements	`<lookup-key>`, `<column>`
Metadata Elements	`<columns>`, `<column>`, `<query>`, `<sql>`, `<parameter>`
Data Elements	`<dataset>`, `<row>`, `<col>`

Copyright © 1999, 2025, Oracle and/or its affiliates.

23.3.3.1 Top-Level Table Element

The top-level table element is described.

Table 23-4 Top-Level Table Element

Tag	Description	Required Attributes	Optional Attributes	Contents
<table>	Corresponds to a single table. It encloses all the other elements of the document.	name	lang, space, normalize-langtag	The order of the elements within <table> is: 1. <translation> (optional) 2. <lookup-key> 3. <columns> 4. <dataset>

23.3.3.2 Translation Elements

The translation elements are described.

Table 23-5 Translation Elements

Tag	Description	Required Attributes	Optional Attributes	Contents
<translation>	Contains generic information pertinent to translation.	None	None	Zero or more <target> elements and zero or more <restype> elements
<target>	Specifies a language to which this document is translated.	None	None	A language identifier as defined by [IETFRFC1766]
<restype>	Declares a type of resource.	name	expansion	Empty element

23.3.3.3 Lookup Key Elements

The lookup key elements are described.

Table 23-6 Lookup Key Elements

Tag	Description	Required Attributes	Optional Attributes	Contents
<lookup-key>	Contains the <column> element(s) based on which the TransX recognizes the rows as identical or duplicate.	name	None	Zero or more <column> elements

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 23-6 (Cont.) Lookup Key Elements

Tag	Description	Required Attributes	Optional Attributes	Contents
<column>	A <column> element under <lookup-key> element indicates a column to be used to recognize the rows as identical or duplicate. Columns with the same values in specified column(s) are considered duplicate, regardless of the values in the other column(s). A lookup key <column> must have corresponding <col>umns in the <dataset> section or be declared as a <column> with a constant expression or a <query> in the <columns> section.	name	None	Empty element

23.3.3.4 Metadata Elements

The metadata elements are described.

Table 23-7 Metadata Elements

Tag	Description	Required Attributes	Optional Attributes	Contents
<columns>	Contains data about the data to be loaded.	None	None	One or more <column> elements
<column>	Specifies a column that corresponds to <col> elements under the <dataset> element. After a <column> is defined the corresponding <col> element must appear in every <row> unless the column has the sequence, constant or query attribute.	name, , type in either order. The recommended sequence is name, type, then optional attributes.	translate, constant, sequence, virtual, useforupdate, maxsize, size-unit, restype in any order	Zero or one <query> or <sql> element
<query>	Specifies a SQL query whose result is used to fill in the column to which this element belongs.	text	None	Zero or more <parameter> elements
<sql>	Specifies a SQL statement whose result, if any, is used to fill in the column to which this element belongs.	text	None	Zero or more <parameter> elements
<parameter>	Specifies a parameter of a <query> element.	id and either col or constant. If col is specified, the referenced column cannot have the query, constant, or sequence attributes.	translate, trans-key	Empty

23.3.3.5 Data Elements

The data elements are <dataset>, <row>, and <col>.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 23-8 describes the data elements.

Table 23-8 Data Elements

Tag	Description	Required Attributes	Optional Attributes	Contents
`<dataset>`	Contains data to be loaded into the database.	None	None	One or more `<row>` elements
`<row>`	Contains data about the data to be loaded `<dataset>` element.	None	None	Zero or more `<col>` elements
`<col>`	Specifies an instance of a piece of data to be loaded to a database column, or for a virtual column, a piece of data to be used to get an actual data to be loaded to a database column.	`name`	`trans-key`	Data for use by applications

23.3.4 Attributes in DLF

The various attributes used in the DLF elements are listed. An attribute is never specified more than once for each element. Along with some of the attributes are the recommended attribute values. Values for these attributes are case-sensitive.

Table 23-9 Attributes

Type of Attribute	Attributes
DLF Attributes	`name`, `type`, `translate`, `constant`, `sequence`, `virtual`, `useforupdate`, `maxsize`, `size-unit`, `restype`, `text`, `id`, `col`, `trans-key`, `translation-note`, `normalize-langtag`
XML Namespace Attributes	`xml:space`

23.3.4.1 DLF Attributes

The DLF attributes are described. These attributes are shared among the DLF elements.

Table 23-10 DLF Attributes

Attribute	Description	Value Description	Default Value	Used by Elements
`lang`	Specifies the language of the document.	This is equivalent to the `xml:lang` attribute. The values of the attribute are language identifiers as defined by [IETFRFC4646]. This attribute does not affect data loading operation in any way.	None; if absent, `"en"` is assumed	`<table>`

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 23-10 (Cont.) DLF Attributes

Attribute	Description	Value Description	Default Value	Used by Elements
normalize-langtag	Specifies how to normalize the case of language tag.	"none", "standard", "uppercase" or "lowercase". The meanings are: none—no normalization. the values in the language column on DLF are used as they are standard—the style as recommended by the standards * lowercase for the 2 letter language code * uppercase for the 2 letter country code * titlecase for the 4 letter script code * lowercase for others uppercase—uppercase everything lowercase—lowercase everything	none	<table>
space	Specifies how white spaces (ASCII spaces, tabs and line-breaks) are treated.	"default" or "preserve" The value "default" signals that applications' default white-space processing modes are acceptable; the value "preserve" indicates the intent that applications preserve all white space. If this intent is declared at the root table element, it is considered to apply to all string data elements in the whole document. If declared at column level, it is considered to apply to the specified column of every row. If this attribute is declared in the <dataset> section, the intent applies only to the immediate text data. Declaration at the document or column level may be overridden with another instance of the space attribute.	"default"	<table>, <column>, <col>
name	Specifies the name of an object such as table, column, restype, and so forth.	String: This is a database table name for the <table> element, and a column name for the <column> or <col> element.	Not applicable	<table>, <column>, <col>

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 23-10 (Cont.) DLF Attributes

Attribute	Description	Value Description	Default Value	Used by Elements
`type`	The data type of a column in the data set. This attribute specifies the kind of text contained in the `<col>` element in the data set. Depending on this type, TransX may apply different processes to the data. Because implicit data type conversion is provided by XSU and Java Database Connectivity (JDBC), TransX does not do its own parsing based on this type information. It uses this attribute to choose appropriate intermediate data types in Java for columns of `date` or `dateTime` type, in which case the standard date formats are accepted.	String: possible values are `"number"`, `"string"`, `"date"`, `"dateTime"` or `"binary"`. The lexical representation of a value of number type must be supplied in the SQL language syntax, no matter what the current locale is. The SQL syntax uses no digit grouping separator (usually comma), but uses a dot as the decimal separator (usually dot).For the binary data type, the data value specified in a text field between the `col` tags indicates the name of a file that contains the actual binary data. Raw data cannot be embedded in the text field.For the other data types (`string`, `date`, and `dateTime`) the representation is constrained by the corresponding types in the XML Schema specification.For simplicity, DLF accepts only standard date formats of XML Schema in the form "CCYY-MM-DDThh:mm:ss" (dateTime) or "CCYY-MM-DD" (date). No other date format is recognized. TransX uses this attribute for: • Bind virtual columns to parameters of a query • Bind the result of a query to a corresponding column	Not applicable	`<column>`
`translate`	Indicates whether to translate the text of this column or parameter.	Either `"yes"` or `"no"`	`"no"`	`<column>`, `<parameter>`
`constant`	Specifies a constant value for this column or parameter.	The value of this column for every row	Not applicable	`<column>`, `<parameter>`
`language`	Specifies language identifier for this column	Language identifier or a placeholder. "%x" gets the value from the `xml:lang` attribute of the root table element.	Not applicable	`<column>`
`sequence`	Specifies a sequence in the database used to fill in the value for this column.	String: The name of a sequence in the database	Not applicable	`<column>`
`virtual`	Indicates that this column provides data used to construct another piece of data, which in turn is loaded into the database. A virtual column does not exist in the database. It is typically used to provide a value of a parameter in a query. A virtual column cannot be a lookup-key column. A virtual column with a query throws the result away.	Either `"yes"` or `"no"`	`"no"`	`<column>`

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 23-10 (Cont.) DLF Attributes

Attribute	Description	Value Description	Default Value	Used by Elements
useforupdate	Indicates whether to use the value of this column for the update when uploading seed data. This attribute has no effect unless TransX is in the mode to update duplicate rows. A virtual column cannot have this attribute set to yes.	Either "yes" or "no". If this attribute is set to "no", then the value of the column remains unchanged on the update operation.	"yes"	<column>
maxsize	Specifies the maximum size for the data for this column.	Numeric value in the unit specified by the size-unit attribute. If this attribute is set and the size-unit is not set, the size is assumed to be in characters.	None	<column>
size-unit	Specifies the unit of size specified in the maxsize attribute.	Units. Recommended values are "char" for characters, "byte" for bytes. For supplemental characters, they take two "char" units.	"char"	<column>
restype	Indicates the type of data contained in this column.	A resource type. The value must match with the name of a <restype> element.	None	<column>
expansion	Indicates the maximum size up to which translated strings are allowed to become longer for this type of resource.	A numeric value in percentage of increase.	0%	<restype>
text	Specifies a SQL query statement to get a value to put in the column to which the query belongs.	A SQL statement. Zero or more parameters can be specified with an identifier preceded by a colon. The statement returns a single row of a single value. Any excessive result is discarded.	Not applicable	<query>
id	Specifies a placeholder used in a SQL query statement with parameters. The value of the column specified by the sibling col attribute is associated as a parameter to the query.	String: an identifier that appears in the text attribute of the parent query element.	Empty string	<parameter>
col	Specifies a column to be associated with a placeholder in the query specified by the sibling id attribute.	String: a column name. The column must be other than the column this attribute is a part of.	Not applicable	<parameter>
trans-key	Specifies a key for translation.	String: a translation key. The value must be unique in a translation domain.	Not applicable	<col>, <parameter>
translation-note	Specifies notes for translation.	String: Translation notes.	Not applicable	<col>, <column>, <parameter>

23.3.4.2 XML Namespace Attributes

The XML namespace attributes are described.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 23-11 XML Namespace Attributes

Attribute	Description	Value Description	Default Value	Used by Elements
`xml:space`	Specifies how white space (ASCII spaces, tabs and line-breaks) are treated.	`"default"` or `"preserve"` The value `"default"` signals that applications' default white space processing modes are acceptable for this element; the value `"preserve"` indicates the intent that applications preserve all the white space. This declared intent is considered to apply to all elements within the content of the element where it is specified, unless overridden with another instance of the `xml:space` attribute.	`"default"`	None
`xml:lang`	Specifies the language of the content.	A language tag defined by RFC 4646.	Not applicable	`table`

23.4 DLF Examples

Topics here include minimal, typical, and localized DLF documents.

23.4.1 Minimal DLF Document

A minimal DLF document is presented.

Example 23-2 Minimal DLF Document

```
<?xml version="1.0" ?>
<table name="dual">
 <lookup-key/>
 <columns>
  <column name="DUMMY" type="string">
 </columns>
 <dataset>
  <row>
   <col name="DUMMY">X</col>
  </row>
 </dataset>
</table>
```

23.4.2 Typical DLF Document

A sample DLF document that contains seed data for table `CLK_STATUS_L` is presented.

Example 23-3 Sample DLF Document

```
<!--
  - $Header: $
  -
  - Copyright (c) 2001 Oracle Corporation. All Rights Reserved.
  -
  - NAME
  -   status.xml - Seed file for the CLK_STATUS_L table
  -
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
  - DESCRIPTION
  -   This file contains seed data for the Status level table.
  -
  - NOTES
  -
  - MODIFIED   (MM/DD/YY)
  -   dchiba    06/11/01 - Adaption to enhancements of data loading tool
  -   dchiba    05/23/01 - Adaption to generic data loading tool
  -   rbolsius  05/07/01 - Created
  -->

<table name="clk_status_l" xml:space="preserve">
  <lookup-key>
    <!--column name="status_id" /-->
    <column name="status_code" />
  </lookup-key>

  <columns>
    <column name="status_id"          type="number" sequence="clk_status_seq"
useforupdate="no"/>
    <column name="status_code"        type="number" />
    <column name="status_name"        type="string" translate="yes" />
    <column name="status_description" type="string" translate="yes" />
    <column name="version_created"    type="number" constant="0" />
    <column name="version_updated"    type="number" constant="0" />
    <column name="status_type_code"   type="string" virtual="yes" />
    <column name="status_type_id"     type="number" >
      <query text="select status_type_id from clk_status_type_l where status_type_code = :1"
>
        <parameter id="1" col="status_type_code" />
      </query>
    </column>
  </columns>

  <dataset>

    <row>
      <col name="status_code" >100</col>
      <col name="status_name" trans-key="stts-name-1" >Continue</col>
      <col name="status_description" trans-key="stts-desc-1" >
        The client should continue with its request.</col>
      <col name="status_type_code" >INFO</col>
    </row>

    <row>
      <col name="status_code" >101</col>
      <col name="status_name" trans-key="stts-name-2" >Switching Protocols</col>
      <col name="status_description" trans-key="stts-desc-2" >
        The server understands and is willing to comply with the client's
        request (via the Upgrade message header field) for a change in the
        application protocol being used on this connection.</col>
      <col name="status_type_code" >INFO</col>
    </row>

    <row>
      <col name="status_code" >200</col>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
      <col name="status_name" trans-key="stts-name-3" >OK</col>
      <col name="status_description" trans-key="stts-desc-3" >
        The request has succeeded.</col>
      <col name="status_type_code" >SUCCESS</col>
    </row>

    <row>
      <col name="status_code" >201</col>
      <col name="status_name" trans-key="stts-name-4" >Created</col>
      <col name="status_description" trans-key="stts-desc-4" >
        The request has been fulfilled and resulted in a new resource being
        created.</col>
      <col name="status_type_code" >SUCCESS</col>
    </row>

    <row>
      <col name="status_code" >202</col>
      <col name="status_name" trans-key="stts-name-5" >Accepted</col>
      <col name="status_description" trans-key="stts-desc-5" >
        The request has been accepted for processing, but the processing has
        not been completed.</col>
      <col name="status_type_code" >SUCCESS</col>
    </row>

    <row>
      <col name="status_code" >203</col>
      <col name="status_name" trans-key="stts-name-6" >Non-Authoritative Information</col>
      <col name="status_description" trans-key="stts-desc-6" >
        The returned metainformation in the entity-header is not the
        definitive set as available from the origin server, but is gathered
        from a local or a third-party copy.</col>
      <col name="status_type_code" >SUCCESS</col>
    </row>

    <row>
      <col name="status_code" >204</col>
      <col name="status_name" trans-key="stts-name-7" >No Content</col>
      <col name="status_description" trans-key="stts-desc-7" >
        The server has fulfilled the request but does not need to return an
        entity-body, and might want to return updated metainformation.</col>
      <col name="status_type_code" >SUCCESS</col>
    </row>

    <!-- ... -->

  </dataset>
</table>
```

23.4.3 Localized DLF Document

An example of elements and attributes for localization is shown.

Example 23-4 DLF with Localization

```
<?xml version="1.0"?>
<table name="table_name" xml:lang="en" xml:space="preserve">
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
<translation>
<target>ar</target>
<target>bs</target>
<target>es</target>
<restype name="alt" expansion="50%"/>
<restype name="foo" expansion="50%"/>
<restype name="bar" expansion="30%"/>
</translation>

<lookup-key><column name="resid" /></lookup-key>

<columns>
<column name="resid" type="number" sequence="seq_foo" useforupdate="no"/>
<column name="image" type="binary"/>
<column name="alt_text" type="string" translate="yes" maxsize="30"
        size-unit="byte" restype="alt"/>

</columns>

<dataset>
<col name="image">foo1.gif</col>
<col name="alt_text">Hello world</col>
</dataset>

</table>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

24

Using the XSQL Pages Publishing Framework

An explanation is given of how to use the basic features of the XSQL pages publishing framework.

Related Topics

- Using the XSQL Pages Publishing Framework: Advanced Topics
  An explanation is given of how to use advanced features of the XSQL pages publishing framework.

24.1 Introduction to the XSQL Pages Publishing Framework

The Oracle XSQL pages publishing framework is an extensible platform for publishing Extensible Markup Language (XML) in multiple formats.

The Java-based XSQL servlet, which is the center of the framework, provides a declarative interface for dynamically publishing dynamic web content based on relational data.

The XSQL framework combines the power of structured query language (SQL), XML, and Extensible Stylesheet Language Transformation (XSLT). You can use it to create declarative templates called XSQL pages to perform these actions:

- Assemble dynamic XML datagrams based on parameterized SQL queries
- Transform datagrams with XSLT to generate a result in an XML, HTML, or text-based format

An XSQL page, so called because its default extension is `.xsql`, is an XML file that contains instructions for the XSQL servlet. The Example 24-1 shows a simple XSQL page. It uses the <xsql:query> action element to query the `hr.employees` table.

You can present a browser client with the data returned from the query in Example 24-1. Assembling and transforming information for publishing requires no programming. You can perform most tasks in a declarative way. If a built-in feature does not fit your needs, however, you can use Java to integrate custom data sources or perform customized server-side processing.

In the XSQL pages framework, the *assembly* of information to be published is separate from presentation. This architectural feature enables you to do this:

- Present the same data in multiple ways, including tailoring the presentation appropriately to the type of client device making the request —browser, cellular phone, personal digital assistant (PDA), and so on.
- Reuse data by aggregating existing pages into new ones
- Revise and enhance the presentation independently of the content

Example 24-1 Sample XSQL Page

```
<?xml version="1.0">
<?xml-stylesheet type="text/xsl" href="emplist.xsl"?>
<xsql:query connection="hr" xmlns:xsql="urn:oracle-xsql">
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
 SELECT * FROM employees
</xsql:query>
```

24.1.1 Prerequisites for Using the XSQL Pages Publishing Framework

Prerequisites for using the XSQL pages publishing framework are described.

This chapter assumes that you are familiar with these technologies:

- Oracle Database SQL. The XSQL framework accesses data in a database.
- Procedural Language/Structured Query Language (PL/SQL). Oracle XML Developer's Kit (XDK) supplies a PL/SQL application programming interface (API) for XML SQL Utility (XSU) that mirrors the Java API.
- Java Database Connectivity (JDBC). The XSQL pages framework depends on a JDBC driver for database connections.
- Extensible Stylesheet Language Transformations (XSLT). You can use XSLT to transform the data into a format appropriate for delivery to the user.
- XML SQL Utility (XSU). The XSQL pages framework uses XSU to query the database.

24.2 Using the XSQL Pages Publishing Framework: Overview

Topics here include basic use, setting up, running the demo programs, and using the command-line utility.

24.2.1 Using the XSQL Pages Framework: Basic Process

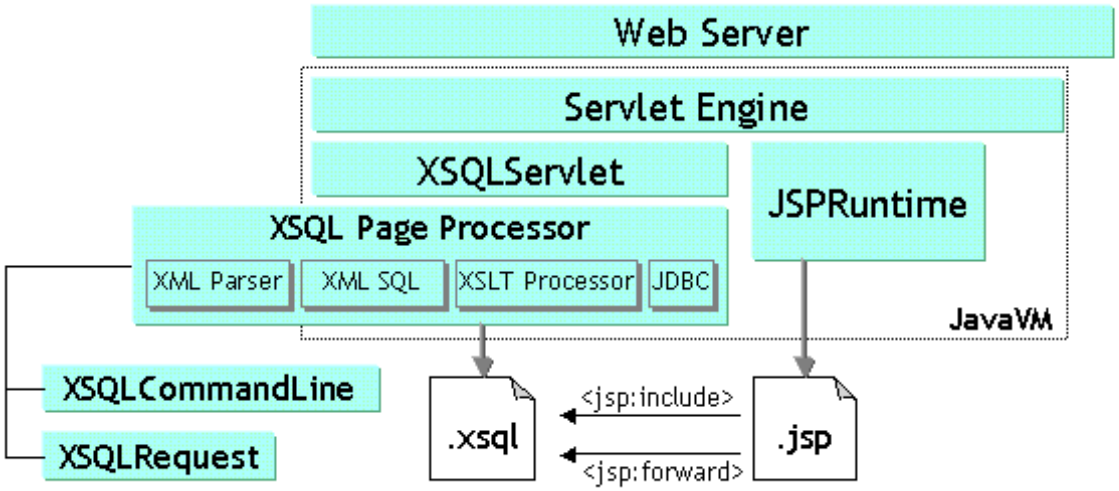
The XSQL page processor engine interprets, caches, and processes the contents of XSQL pages. Basic use of the XSQL pages framework is described.

Figure 24-1 shows the basic architecture of the XSQL pages publishing framework. The XSQL page processor provides access from this entry points:

- From the command line or in batch mode with the XSQL command-line utility. The `oracle.xml.xsql.XSQLCommandLine` class is the command-line interface.
- Over the web by using the XSQL servlet installed in a web server. The `oracle.xml.xsql.XSQLServlet` class is the servlet interface.
- As part of JSP applications by using `<jsp:include>` to include a template or `<jsp:forward>` to forward a template.
- Programmatically by using the `oracle.xml.xsql.XSQLRequest` Java class.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Figure 24-1 XSQL Pages Framework Architecture



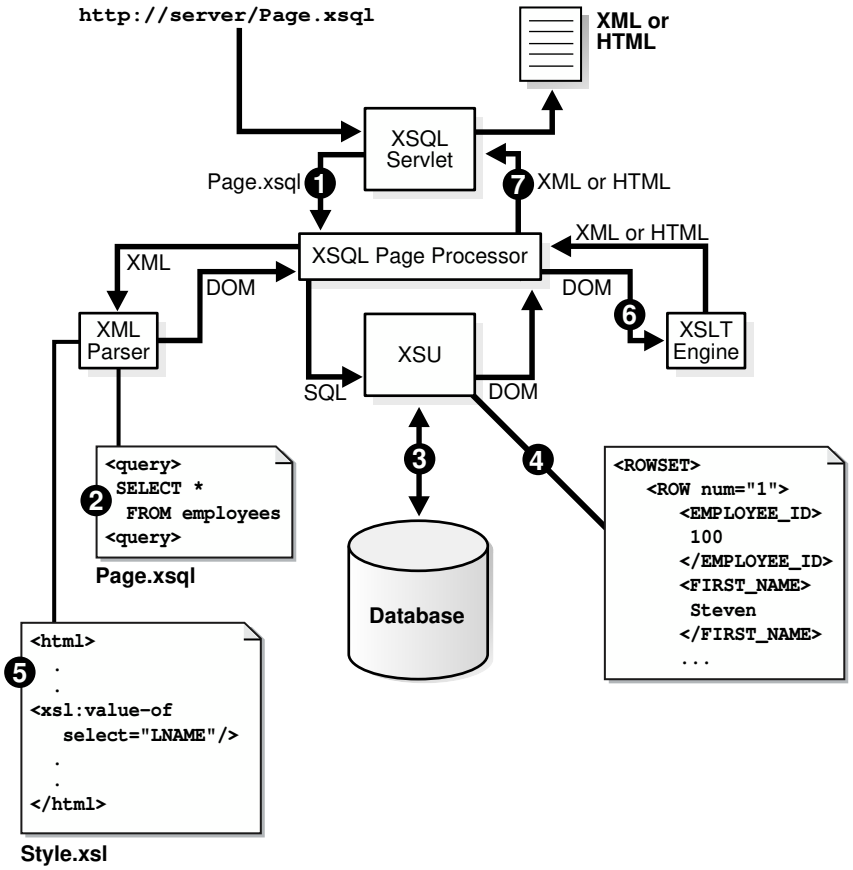
You can run the same XSQL pages from any of the access points shown in Figure 24-1. Regardless of which way you use the XSQL page processor, it performs these actions to generate a result:

1. Receives a request to process an XSQL page. The request can come from the command-line utility or programmatically from an `XSQLRequest` object.
2. Assembles an XML datagram by using the result of one or more SQL queries. The query is specified in the <xsql:query> element of the XSQL page.
3. Returns this XML datagram to the requester.
4. Optionally transforms the datagram into any XML, HTML, or text-based format.

Figure 24-2 shows a typical web-based scenario in which a web server receives an HTTP request for `Page.xsql`, which contains a reference to the XSLT stylesheet `Style.xsl`. The XSQL page contains a database query.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Figure 24-2 Web Access to XSQL Pages



The XSQL page processor shown in Figure 24-2 performs these steps:

1. Receives a request from the XSQL Servlet to process `Page.xsql`.
2. Parses `Page.xsql` with the Oracle XML Parser and caches it.
3. Connects to the database based on the value of the connection attribute on the document element.
4. Generates the XML datagram by replacing each XSQL action element, for example, <xsql:query>, with the XML results returned by its built-in action handler.
5. Parses the `Style.xsl` stylesheet and caches it.
6. Transforms the datagram by passing it and the `Style.xsl` stylesheet to the Oracle XSLT processor.
7. Returns the resulting XML or HTML document to the requester.

During the transformation step in this process, you can use stylesheets that conform with the W3C XSLT 1.0 or 2.0 standard to transform the assembled datagram into document formats such as:

- 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

Copyright © 1999, 2025, Oracle and/or its affiliates.

- Text documents such as e-mails, SQL scripts, Java programs, and so on
- Arbitrary XML-based document formats

24.2.2 Setting Up the XSQL Pages Framework

You can develop and use XSQL pages in various scenarios.

24.2.2.1 Creating and Testing XSQL Pages with Oracle JDeveloper

The following Oracle JDeveloper tasks are covered here: creating an XSQL page, adding XSQL action elements to an XSQL page, checking the syntax of an XSQL page, testing an XSQL page, and adding an XSQL runtime library to your project library list so that environment variable `CLASSPATH` is properly set.

The IDE supports these features:

- Color-coded syntax highlighting
- XML syntax checking
- In-context drop-down lists that help you pick valid XSQL tag names and auto-complete tag and attribute names
- XSQL page deployment and testing
- Debugging tools
- Wizards for creating XSQL actions

To create an XSQL page in an Oracle JDeveloper project:

1. Create or open a project.
2. Select **File** and then **New**.
3. In the **New Gallery** dialog box, select the **General** category and then **XML**.
4. In the **Item** window, select **XSQL Page** and click **OK**. Oracle JDeveloper loads a tab for the new XSQL page into the central window.

To add XSQL action elements such as <xsql:query> to your XSQL page, place the cursor where you want the new element to go and click an item in the Component Palette. A wizard opens that takes you through the steps of selecting which XSQL action you want to use and which attributes you must provide.

To check the syntax of an XSQL page, place the cursor in the page and right-click **Check XML Syntax**. If there are any XML syntax errors, Oracle JDeveloper displays them.

To test an XSQL page, select the page in the navigator and right-click **Run**. Oracle JDeveloper automatically starts a local web server, properly configured to run XSQL pages, and tests your page by starting your default browser with the appropriate URL to request the page. After you have run the XSQL page, you can continue to make modifications to it in the IDE. And, you can modify any XSLT stylesheets with which it might be associated. After saving the files in the IDE, you can immediately refresh the browser to observe the effect of the changes.

You must add the XSQL runtime library to your project library list so that the `CLASSPATH` is properly set. The IDE adds this entry automatically when you go through the New Gallery dialog to create a new XSQL page, but you can also add it manually to the project as follows:

1. Right-click the project in the Applications Navigator.
2. Select **Project Properties**.

Copyright © 1999, 2025, Oracle and/or its affiliates.

3. Select **Profiles** and then **Libraries** from the navigation tree.
4. Move **XSQL Runtime** from the **Available Libraries** pane to **Selected Libraries**.

24.2.2.2 Setting the CLASSPATH for XSQL Pages

Outside of the Oracle JDeveloper environment, you must ensure that the XSQL page processor engine is properly configured.

Ensure that the appropriate Java Archive (JAR) files are in the `CLASSPATH` of the Java Virtual Machine (JVM) that processes the XSQL Pages. The complete set of XDK JAR files is described in Table 11-1. The JAR files for the XSQL framework include:

- `xml.jar`, the XSQL page processor
- `xmlparserv2.jar`, the Oracle XML parser
- `xsu12.jar`, the Oracle XML SQL utility (XSU)
- `ojdbc6.jar`, the Oracle JDBC driver

> **Note**
>
> The XSQL servlet can connect to any database that has Java Database Connectivity (JDBC) support. Indicate the appropriate JDBC driver class and connection URL in the XSQL configuration file connection definition. Object-relational functionality works only when using Oracle Database with the Oracle JDBC driver.

If you have configured your `CLASSPATH` as instructed in Setting Up the XDK for Java Environment, you need to add the *directory* only where the XSQL pages configuration file resides. In the database installation of XDK, the directory for `XSQLConfig.xml` is `$ORACLE_HOME/xdk/admin`.

On Windows your `%CLASSPATH%` variable contains these entries:

```
%ORACLE_HOME%\lib\ojdbc6.jar;%ORACLE_HOME%\lib\xmlparserv2.jar;
%ORACLE_HOME%\lib\xsu12.jar;C:\xsql\lib\xml.jar;%ORACLE_HOME%\xdk\admin
```

On UNIX the `$CLASSPATH` variable contains these entries:

```
$ORACLE_HOME/lib/ojdbc6.jar:$ORACLE_HOME/lib/xmlparserv2.jar:
$ORACLE_HOME/lib/xsu12.jar:$ORACLE_HOME/lib/xml.jar:$ORACLE_HOME\xdk\admin
```

> **Note**
>
> If you are deploying your XSQL pages in a Java Platform, Enterprise Edition (Java EE) web application archive (WAR) file, then you can include the XSQL JAR files in the `./WEB-INF/lib` directory of the WAR file.

24.2.2.3 Configuring the XSQL Servlet Container

You can install the XSQL servlet in a variety of different web servers. See the file `$ORACLE_HOME/xdk/readme.html` for servlet installation instructions.

Copyright © 1999, 2025, Oracle and/or its affiliates.

24.2.2.4 Setting Up the Connection Definitions

XSQL pages specify database connections by using a short name for a connection that is defined in the XSQL configuration file, which by default is named `$ORACLE_HOME/xdk/admin/XSQLConfig.xml`.

> **Note**
>
> If you are deploying your XSQL pages in a Java EE WAR file, then you can place the `XSQLConfig.xml` file in the `./WEB-INF/classes` directory of your WAR file.

The sample XSQL page shown in Example 24-1 contains this connection information:

```
<xsql:query connection="hr" xmlns:xsql="urn:oracle-xsql">
```

Connection names are defined in the `<connectiondefs>` section of the XSQL configuration file. Example 24-2 shows the relevant section of the sample configuration file included with the database, with the `hr` connection in bold.

For each database connection, you can specify these elements:

- `<username>`, the database user name
- `<password>`, the database password
- `<dburl>`, the JDBC connection string
- `<driver>`, the fully qualified class name of the JDBC driver to use
- `<autocommit>`, which optionally forces `AUTOCOMMIT` to `TRUE` or `FALSE`

Specify an `<autocommit>` child element to control the setting of the JDBC autocommit for any connection. If no `<autocommit>` child element is set for a `<connection>`, then the autocommit setting is not set by the XSQL connection manager. In this case, the setting is the default autocommit setting for the JDBC driver.

You can place an arbitrary number of `<connection>` elements in the XSQL configuration file to define your database connections. 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**
>
> The `XSQLConfig.xml` file contains sensitive database user name and password information that must be kept secure on the database server. See Security Considerations for XSQL Pages for instructions.

Example 24-2 Connection Definitions Section of XSQLConfig.xml

```
<connectiondefs>
  ...
  <connection name="hr">
    <username>hr</username>
    <password>hr_password</password>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
    <dburl>jdbc:oracle:thin:@localhost:1521:ORCL</dburl>
    <driver>oracle.jdbc.driver.OracleDriver</driver>
    <autocommit>false</autocommit>
  </connection>
  ...
</connectiondefs>
```

24.2.3 Running the XSQL Pages Demo Programs

Demo programs for the XSQL servlet are included in `$ORACLE_HOME/xdk/demo/java/xsql`.

Table 24-1 lists the demo subdirectories and explains the included demos. The Demo Name column refers to the title of the demo listed on the XSQL Pages & XSQL Servlet home page. Running the XSQL Demos explains how to access the home page.

Table 24-1 XSQL Servlet Demos

Directory	Demo Name	Description
`home/`	XSQL Pages & XSQL Servlet	Contains the pages that display the tabbed home page of the XSQL demos and the online XSQL help that you can access from that page. As explained in Running the XSQL Demos. you can invoke the XSQL home page from the `index.html` page.
`helloworld/`	Hello World Page	Shows the simplest possible XSQL page.
`emp/`	Employee Page	XSQL page showing XML data from the `hr.employees` table, using XSQL page parameters to control what employees are returned and which columns to use for the database sort. Uses an associated XSLT stylesheet to format the results as an HTML Form containing the `emp.xsql` page as the form action so the user can refine the search criteria.
`insclaim/`	Insurance Claim Page	Demonstrates several sample queries over the richly structured Insurance Claim object view. The `insclaim.sql` scripts sets up the `INSURANCE_CLAIM_VIEW` object view and populates it with sample data.
`classerr/`	Invalid Classes Page	Uses `invalidclasses.xsl` to format a "live" list of current Java class compilation errors in your schema. The accompanying SQL script sets up the `XSQLJavaClassesView` object view used by the demo. The master/detail information from the object view is formatted into HTML by the `invalidclasses.xsl` stylesheet in the server.
`doyouxml/`	Do You XML? Site	Shows how a simple, data-driven web site can be built with an XSQL page that uses SQL, XSQL substitution variables in the queries, and XSLT for formatting the site. Demonstrates using substitution parameters in both the body of SQL query statements within <xsql:query> tags, and also within the attributes to `<xsql:query>` tags to control behavior such as how many records to display and to skip (for "paging" through query results in a stateless way).
`empdept/`	Emp/Dept Object Demo	Demonstrates how to use an object view to group master/detail information from two existing flat tables such as `scott.emp` and `scott.dept`. The `empdeptobjs.sql` script creates the object view and also the `INSTEAD OF INSERT` triggers that enable the master/detail view to be used as an insert target of `xsql:insert-request`. The `empdept.xsl` stylesheet shows a form of an XSLT stylesheet that looks just like an HTML page without the extra `xsl:stylesheet` or `xsl:transform` at the top. Using a Literal Result Element as a stylesheet is part of the XSLT 1.0 specification. The stylesheet also shows how to generate an HTML page that includes `<link rel="stylesheet">` to enable the generated HTML to fully leverage cascading stylesheets (CSS) for centralized HTML style information, found in the `coolcolors.css` file.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 24-1 (Cont.) XSQL Servlet Demos

Directory	Demo Name	Description
`airport/`	Airport Code Validation	Returns a datagram of information about airports based on their three-letter codes and uses `<xsql:no-rows-query>` 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. The `airport.htm` page shows how to use the XML results of the `airport.xsql` page from a web page with JavaScript to exploit built-in Document Object Model (DOM) functionality in Internet Explorer. When you enter the three-letter airport code on the web page, a JavaScript fetches an XML datagram from XSQL servlet. The datagram corresponds to the code that you entered. If the return indicates no match, then the program collects a "picklist" of possible matches based on information returned in the XML datagram from XSQL servlet
`airport/`	Airport Code Display	Demonstrates use of the same XSQL page as the Airport Code Validation example but supplies an XSLT stylesheet name in the request. This behavior causes the airport information to be formatted as an HTML form instead of being returned as raw XML.
`airport/`	Airport Soap Service	Demonstrates returning airport information as a Simple Object Access Protocol (SOAP) Service.
`adhocsql/`	Adhoc Query Visualization	Demonstrates how to pass a SQL query and an XSLT stylesheet as parameters to the server.
`document/`	XML Document Demo	Demonstrates inserting XML documents into relational tables. The `docdemo.sql` script creates a user-defined type called XMLDOCFRAG containing an attribute of type character large object (CLOB). Try inserting the text of the document in `./xsql/demo/xml99.xml` and providing the name `xml99.xsl` as the stylesheet, and `./xsql/demo/JDevRelNotes.xml` with the stylesheet `relnotes.xsl`. The `docstyle.xsql` page shows an example of the <xsql:include-xsql> action element to include the output of the `doc.xsql` page into its own page before transforming the final output using a client-supplied stylesheet name. The demo uses the client-side XML features of Internet Explorer 5.0 to check the document for well-formedness before allowing it to be posted to the server.
`insertxml/`	XML Insert Request Demo	Demonstrates posting XML from a client to an XSQL page that handles inserting the posted XML data into a database table using the <xsql:insert-request> action element. The demo accepts XML documents in the `moreover.com` XML-based news format. In this case, the program doing the posting of the XML is a client-side web page using Internet Explorer 5.0 and the `XMLHttpRequest` object from JavaScript. If you look at the source for the `insertnewsstory.xsql` page, you'll see it's specifying a table name and an XSLT Transform name. The `moreover-to-newsstory.xsl` stylesheet transforms the incoming XML information into the canonical format that the `OracleXMLSave` utility knows how to insert. Try copying and pasting the example `<article>` element several times within the `<moreovernews>` element to insert several new articles in one shot. The `newsstory.sql` script 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, and so on.
`svg/`	Scalable Vector Graphics Demo	The `deptlist.xsql` page displays a simple list of departments with hyperlinks to the `SalChart.xsql` page. The `SalChart.xsql` page queries employees for a given department passed in as a parameter and uses the associated `SalChart.xsql` stylesheet to format the result into a Scalable Vector Graphics drawing, a bar chart comparing salaries of the employees in that department.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 24-1 (Cont.) XSQL Servlet Demos

Directory	Demo Name	Description
`fop/`	PDF Demo	The `emptable.xsql` page displays a simple list of employees. The `emptable.xsl` 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.
`cursor/`	Cursor Demo	Contains an example of using a nested `CURSOR` expression, which is one of three ways to use the default <xsql:query> element to produce nested elements.
`actions/`		Contains the source code for two example custom actions.

24.2.3.1 Setting Up the XSQL Demos

How to set up the XSQL demos is described.

1. Change into the `$ORACLE_HOME/xdk/demo/java/xsql` directory (UNIX) or `%ORACLE_HOME%\xdk\demo\java\xsql` directory (Windows).
2. Start SQL*Plus and connect to your database as `ctxsys`—the schema owner for the Oracle Text packages—and issue this statement:

   ```
   GRANT EXECUTE ON ctx_ddl TO scott;
   ```

3. Connect to your database as a user with `DBA` privileges and issue this statement:

   ```
   GRANT QUERY REWRITE TO scott;
   ```

   The preceding query enables `scott` to create a function-based index that one of the demos requires to perform case-insensitive queries on descriptions of airports.
4. Connect to your database as `scott`. You are prompted for the password.
5. Run the SQL script `install.sql` in the current directory. This script runs all SQL scripts for all the demos:

   ```
   @install.sql
   ```

6. Change to the `./doyouxml` subdirectory, and run this command to import sample data for the "Do You XML?" demo (you are prompted for the password):

   ```
   imp scott file=doyouxml.dmp
   ```

7. To run the Scalable Vector Graphics (SVG) demonstration, install an SVG plug-in such as Adobe SVG plug-in into your browser.

24.2.3.2 Running the XSQL Demos

The XSQL demos are designed to be accessed through a web browser.

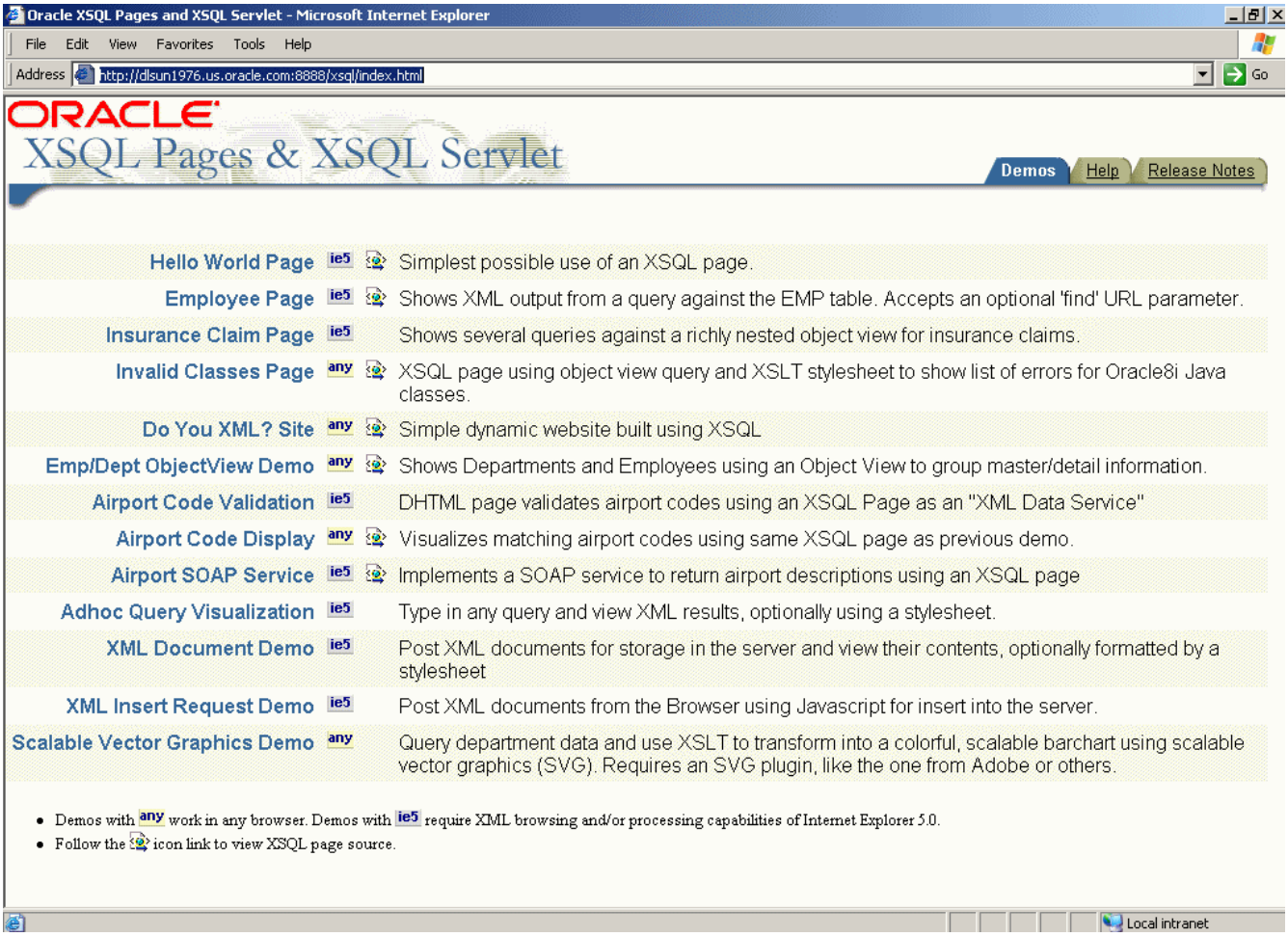
If you have set up the XSQL servlet in a web server as described in Configuring the XSQL Servlet Container, then you can access the demos through this URL, substituting appropriate values for `yourserver` and `port`:

```
http://yourserver:port/xsql/index.html
```

Figure 24-3 shows a section of the XSQL home page in Internet Explorer. (You must use browser version 5 or later.)

Copyright © 1999, 2025, Oracle and/or its affiliates.

Figure 24-3 XSQL Home Page



The demos are designed to be self-explanatory. Click the demo titles—**Hello World Page**, **Employee Page**, and so forth—and follow the online instructions.

24.2.4 Using the XSQL Pages Command-Line Utility

XDK includes a command-line Java interface that runs the XSQL page processor. You can process any XSQL page with the XSQL command-line utility.

Often the content of a dynamic page is based on data that does not frequently change. To optimize performance of your web publishing, you can use operating system facilities to schedule offline processing of your XSQL pages. This technique enables the processed results to be served statically by your web server.

The `$ORACLE_HOME/xdk/bin/xsql` and `%ORACLE_HOME%\xdk\bin\xsql.bat` shell scripts run the `oracle.xml.xsql.XSQLCommandLine` class. Before invoking the class ensure that your environment is configured as described in Setting Up the XSQL Pages Framework. Depending on how you invoke the utility, the syntax is either of these:

```
java oracle.xml.xsql.XSQLCommandLine xsqlpage [outfile] [param1=value1 ...]
xsql xsqlpage [outfile] [param1=value1 ...]
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

If you specify an *outfile*, then the result of processing xsqlpage is written to it; otherwise the result goes to standard out. You can pass any number of parameters to the XSQL page processor, which are available for reference by the XSQL page processed as part of the request. However, these parameter names are recognized by the command-line utility and have a predefined behavior:

- xml-stylesheet=stylesheetURL

  Provides the relative or absolute URL for a stylesheet to use for the request. You can also set it to the string none to suppress XSLT stylesheet processing for debugging.

- 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

  Simulates a particular HTTP User-Agent string from the command line so that an appropriate stylesheet for that User-Agent type is selected as part of command-line processing of the page.

24.3 Generating and Transforming XML with XSQL Servlet

The basic tasks that you can perform with your server-side XSQL page templates are described.

24.3.1 Composing XSQL Pages

You can serve database information in XML format over the web with XSQL pages.

For example, suppose your aim is to serve a real-time XML datagram from Oracle of all available flights landing today at JFK airport. Example 24-3 shows a sample XSQL page in a file named AvailableFlightsToday.xsql.

The XSQL page is an XML file that contains any mix of static XML content and XSQL action elements. The file can have any extension, but .xsql is the default extension for XSQL pages. You can modify your servlet engine configuration settings to associate other extensions by using the same technique described in Configuring the XSQL Servlet Container. The servlet extension mapping is configured inside the ./WEB-INF/web.xml file in a Java EE WAR file.

The XSQL page in Example 24-3 begins with this declaration:

```
<?xml version="1.0"?>
```

The first, outermost element in an XSQL page is the **document element**. AvailableFlightsToday.xsql contains a single XSQL action element <xsql:query>, but no static XML elements. In this case the <xsql:query> element is the document element. Example 24-3 represents the simplest useful XSQL page: one that contains a single query. The results of the query replace the <xsql:query> section in the XSQL page.

> **Note**
>
> XSQL Pages Reference describes the complete set of built-in action elements.

Copyright © 1999, 2025, Oracle and/or its affiliates.

The <xsql:query> action element includes an xmlns attribute that declares the xsql namespace prefix as a synonym for the urn:oracle-xsql value, which is the Oracle XSQL namespace identifier:

```
<xsql:query connection="demo" bind-params="City" xmlns:xsql="urn:oracle-xsql">
```

The element also contains a connection attribute whose value is the name of a predefined connection in the XSQL configuration file:

```
<xsql:query connection="demo" bind-params="City" xmlns:xsql="urn:oracle-xsql">
```

The details concerning the user name, password, database, and JDBC driver to be used for the demo connection are centralized in the configuration file.

To include more than one query on the page, you can invent an XML element to wrap the other elements. Example 24-4 shows this technique.

In Example 24-4, the connection attribute and the xsql namespace declaration always go on the document element, whereas the bind-params is specific to the <xsql:query> action.

Example 24-3 Sample XSQL Page in 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 "?" represents a bind variable bound */
  ORDER BY  ExpectedTime      /* to the value of the City parameter.      */
</xsql:query>
```

Example 24-4 Wrapping the <xsql:query> Element

```
<?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 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>
```

24.3.1.1 Using Bind Parameters

The use of bind parameters is described.

The <xsql:query> element shown in Example 24-3 contains a bind-params attribute that associates the values of parameters in the request to bind variables in the SQL statement included in the <xsql:query> tag. The bind parameters in the SQL statement are represented by question marks.

You can use SQL bind variables to parameterize the results of any of the actions in Table 33-1 that allow SQL statements. Bind variables enable your XSQL page template to produce results based on the values of parameters passed in the request.

Copyright © 1999, 2025, Oracle and/or its affiliates.

To use a bind variable, include a question mark anywhere in a statement where bind variables are allowed by SQL. Whenever a SQL statement is executed in the page, the XSQL engine binds the parameter values to the variable by specifying the `bind-params` attribute on the action element.

Example 24-5 shows an XSQL page that binds the bind variables to the value of the `custid` parameter in the page request.

The XML data for a customer with ID of 101 can then be requested by passing the customer id parameter in the request:

```
http://yourserver.com/fin/CustomerPortfolio.xsql?custid=1001
```

The value of the `bind-params` attribute is a space-delimited list of parameter names. The left-to-right order indicates the positional bind variable to which its value is bound in the statement. Thus, if your SQL statement contains five question marks, then the `bind-params` attribute needs a space-delimited list of five parameter names. If the same parameter value must be bound to several different occurrences of a bind variable, then repeat the name of the parameters in the value of the `bind-params` attribute at the appropriate position. Failure to include the same number of parameter names in the `bind-params` attribute as in the query causes an error when the page is executed.

You can use variables in any action that expects a SQL statement or PL/SQL block. The page shown in Example 24-6 shows this technique. The XSQL page contains three action elements:

- <xsql:dml> binds `useridCookie` to an argument in the `log_user_hit` procedure.
- <xsql:query> binds parameter `custid` to a variable in a `WHERE` clause.
- <xsql:include-owa> binds parameters `custid` and `userCookie` to two arguments in the `historical_data` procedure.

Example 24-5 Bind Variables in CustomerPortfolio.xsql

```
<portfolio connnection="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>
```

Example 24-6 Bind Variables with Action Elements in CustomerPortfolio.xsql

```
<portfolio connnection="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>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
  </analysis>
</portfolio>
```

24.3.1.2 Using Lexical Substitution Parameters

For any XSQL action element, you can substitute a lexical substitution parameter for the value of any attribute or the text of any contained SQL statement. Thus, you can parameterize how actions behave and substitute parts of the SQL statements that they perform.

Lexical substitution parameters are referenced with this syntax: `{@ParameterName}`. Example 24-7 shows how you can use two lexical substitution parameters. One parameter in the <xsql:query> element sets the maximum number of rows to be passed in, whereas the other controls the list of columns to be ordered.

Example 24-7 also contains two bind parameters: `dev` and `prod`. For example, you might want to get the open bugs for developer `yxsmith` against product `817`. And, you want to retrieve only 10 rows and order them by bug number. You can fetch the XML for the bug list by specifying parameter values:

```
http://server.com/bug/DevOpenBugs.xsql?
dev=yxsmith&prod=817&max=10&orderby=bugno
```

You can also use the XSQL command-line utility to make the request:

```
xsql DevOpenBugs.xsql dev=yxsmith prod=817 max=10 orderby=bugno
```

Lexical parameters also enable you to specify parameters for the XSQL pages connection and the stylesheet used to process the page. Example 24-8 shows this technique. You can switch between stylesheets `test.xsql` and `prod.xsl` by specifying the name/value pairs `sheet=test` and `sheet=prod`.

Example 24-7 Lexical Substitution Parameters for Rows and Columns in DevOpenBugs.xsql

```
<!-- 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>
```

Example 24-8 Lexical Substitution Parameters for Connections and Stylesheets in DevOpenBugs.xsql

```
<?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
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
      FROM bug_table
     WHERE programmer_assigned = UPPER(?)
       AND product_id          = ?
       AND status < 80
    ORDER BY {@orderby}
  </xsql:query>
</open-bugs>
```

24.3.1.3 Providing Default Values for Bind and Substitution Parameters

You may want to provide a default value for a bind variable or a substitution parameter directly in a page. In this way, the page is parameterized without requiring the requester to explicitly pass in all values in each request.

To include a default value for a parameter, 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, then the XSQL page processor searches for an attribute by the same name on the current action element. If it does not 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.

The page in Example 24-9 defaults the value of the `max` parameter to `10` for both `<xsql:query>` actions in the page.

This page in Example 24-10 defaults the first query to a `max` of `5`, the second query to a `max` of `7`, and the third query to a `max` of `10`.

All defaults are overridden if a value of `max` is supplied in the request, as shown in this example:

```
http://yourserver.com/example.xsql?max=3
```

Bind variables respect the same defaulting rules. Example 24-11 shows how you can set the `val` parameter to `10` by default.

If the page in Example 24-11 is requested without any parameters, it returns this XML datagram:

```
<example>
  <rowset>
    <row>
      <somevalue>10</somevalue>
    </row>
  </row>
</example>
```

Alternatively, assume that the page is requested with this URL:

```
http://yourserver.com/example.xsql?val=3
```

The preceding URL returns this datagram:

```
<example>
  <rowset>
    <row>
      <somevalue>3</somevalue>
    </row>
  </row>
</example>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

You can remove the default value for the `val` parameter from the page by removing the `val` attribute. Example 24-12 shows this technique.

A URL request for the page that does not supply a name/value pair returns this datagram:

```
<example>
  <rowset/>
</example>
```

A bind variable that is bound to a parameter with *neither* a default value *nor* a value supplied in the request is bound to NULL, which causes the `WHERE` clause in Example 24-12 to return no rows.

Example 24-9 Setting a Default Value

```
<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>
```

Example 24-10 Setting Multiple Default Values

```
<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>
```

Example 24-11 Defaults for Bind Variables

```
<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>
```

Example 24-12 Bind Variables with No Defaults

```
<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>
```

24.3.1.4 How the XSQL Page Processor Handles Different Types of Parameters

XSQL pages can make use of parameters supplied in the request and also of page-private parameters. The names and values of page-private parameters 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, then the value of the `param` parameter is resolved in this order:

1. The value of the page-private parameter named `param`, if set
2. The value of the request parameter named `param`, if supplied

Copyright © 1999, 2025, Oracle and/or its affiliates.

3. The default value provided by an attribute named `param` on the current action element or one of its ancestor elements
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, you can also set and reference the HTTP-Session-level variables and HTTP Cookies parameters.

For XSQL pages processed through the XSQL servlet, the value of a parameter `param` is resolved in this order:

1. The value of the page-private parameter `param`, if set
2. The value of the cookie named `param`, if set
3. The value of the session variable named `param`, if set
4. The value of the request parameter named `param`, if supplied
5. The default value provided by an attribute named `param` on the current action element or one of its ancestor elements
6. The value NULL for bind variables and the empty string for lexical parameters

The resolution order means that users cannot supply parameter values in a request to override parameters of the same name set in the HTTP session. Also, users cannot set them as cookies that persist across browser sessions.

24.3.2 Producing Datagrams from SQL Queries

How to produce datagrams using SQL queries is described.

With XSQL servlet properly installed on your web server, you can access XSQL pages by following these basic steps:

1. Copy an XSQL file to a directory under the virtual hierarchy of your web server. Example 24-3 shows the sample page `AvailableFlightsToday.xsql`.

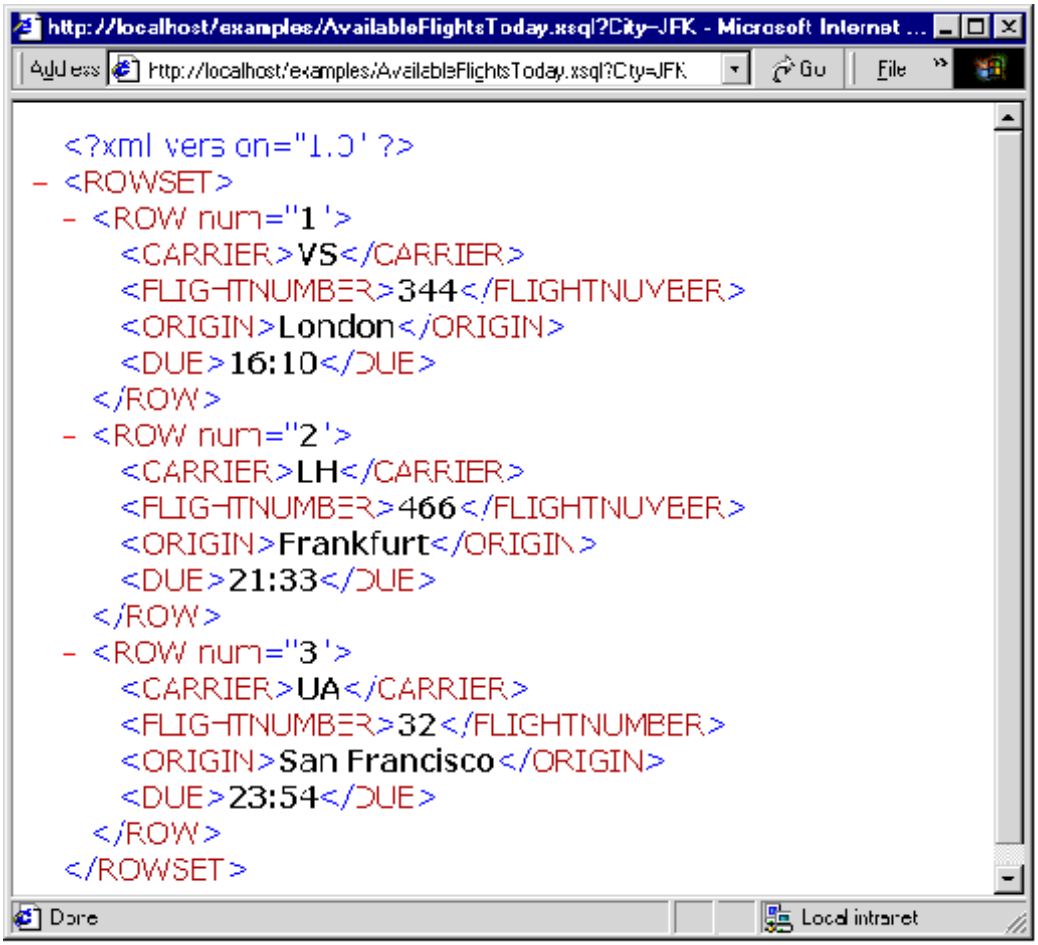
   You can also deploy XSQL pages in a standard Java EE WAR file, which occurs when you use Oracle JDeveloper to develop and deploy your pages to Oracle WebLogic Server.
2. Load the page in your browser. For example, if the root URL is `yourcompany.com`, then you can access the `AvailableFlightsToday.xsql` page through a web browser by requesting this URL:

   ```
   http://yourcompany.com/AvailableFlightsToday.xsql?City=JFK
   ```

The XSQL page processor automatically materializes the results of the query in your XSQL page as XML and returns them to the requester. Typically, another server program requests this XML-based datagram for processing, but if you use a browser such as Internet Explorer then you can directly view the XML result, as shown in Figure 24-4.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Figure 24-4 XML Result from XSQL Page (AvailableFlightsToday.xsql) Query



24.3.3 Transforming XML Datagrams into an Alternative XML Format

If the canonical `<ROWSET>` and `<ROW>` XML output format is not the XML format you need, you can associate an XSLT stylesheet with your XSQL page. The stylesheet can transform the XML datagram in the server before returning the data.

The canonical output is presented in Figure 24-4.

When exchanging data with another program, you typically agree on a document type definition (DTD) that describes the XML format for the exchange. Assume that you are given the `flight-list.dtd` definition and are told to produce your list of arriving flights in a format compliant with the DTD. You can use a visual tool such as XML Authority to browse the structure of the flight-list DTD, as shown in Figure 24-5.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Figure 24-5 Exploring flight-list.dtd with XML Authority

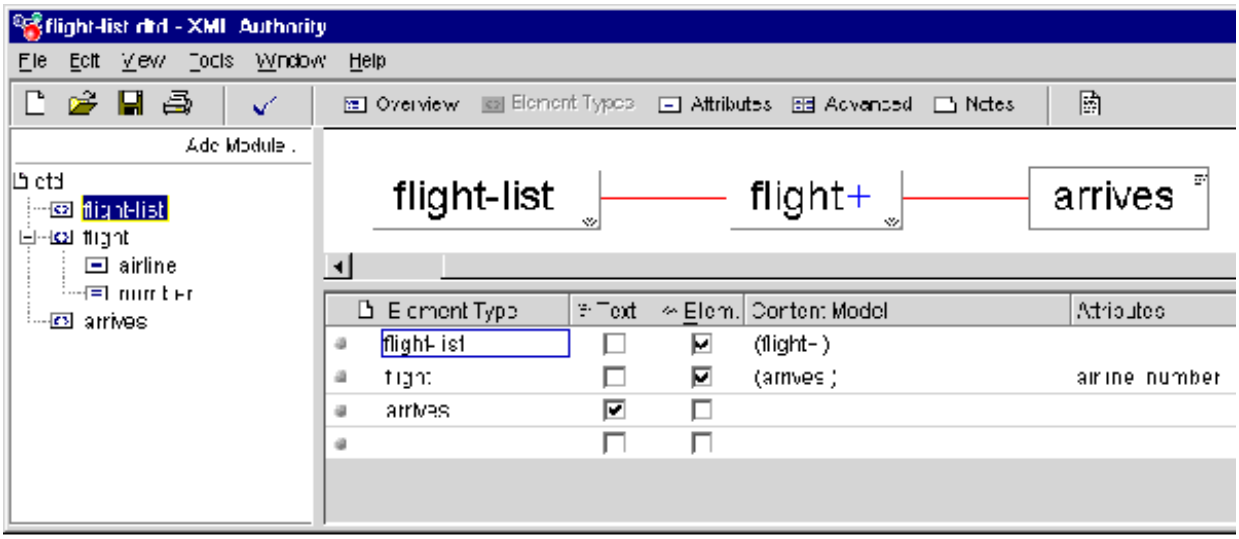


Figure 24-5 shows that the standard XML formats for flight lists are:

- `<flight-list>` element, which contains one or more `<flight>` elements
- `<flight>` elements, which have attributes `airline` and `number`, and each of which contains an `<arrives>` element
- `<arrives>` elements, which contains text

Example 24-13 shows the XSLT stylesheet `flight-list.xsl`. By associating the stylesheet with the XSQL page, you can change the default `<ROWSET>` and `<ROW>` format into the industry-standard `<flight-list>` and `<flight>`.

The XSLT stylesheet is a template that includes the literal elements to produce in the resulting document, such as `<flight-list>`, `<flight>`, and `<arrives>`, interspersed with XSLT actions that enable you to do this:

- Loop over matching elements in the source document with `<xsl:for-each>`
- Plug in the values of source document elements where necessary with `<xsl:value-of>`
- Plug in the values of source document elements into attribute values with the *`{some_parameter}`* notation

The following items have been added to the top-level `<flight-list>` element in the Example 24-13 stylesheet:

- `xmlns:xsl="http://www.w3.org/1999/XSL/Transform"`

  This attribute defines the XML namespace named `xsl` and identifies the URL 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 defined in the XSLT specification. When the namespace is defined, you can use the `<xsl:XXX>` action elements in the stylesheet to loop and plug values in where necessary.

- `xsl:version="1.0"`

Copyright © 1999, 2025, Oracle and/or its affiliates.

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.

You can associate the `flight-list.xsl` stylesheet with the `AvailableFlightsToday.xsql` in Example 24-3 by adding an `<?xml-stylesheet?>` instruction to the top of the page. Example 24-14 shows this technique.

Associating an XSLT stylesheet with the XSQL page causes the requesting program or browser to view the XML in the format as specified by `flight-list.dtd` you were given. Figure 24-6 shows a sample browser display.

Figure 24-6 XSQL Page Results in XML Format

Example 24-13 Industry Standard Formats in flight-list.xsl

```
<!-- 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>
```

Example 24-14 Stylesheet Association in flight-list.xsl

```
<?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
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

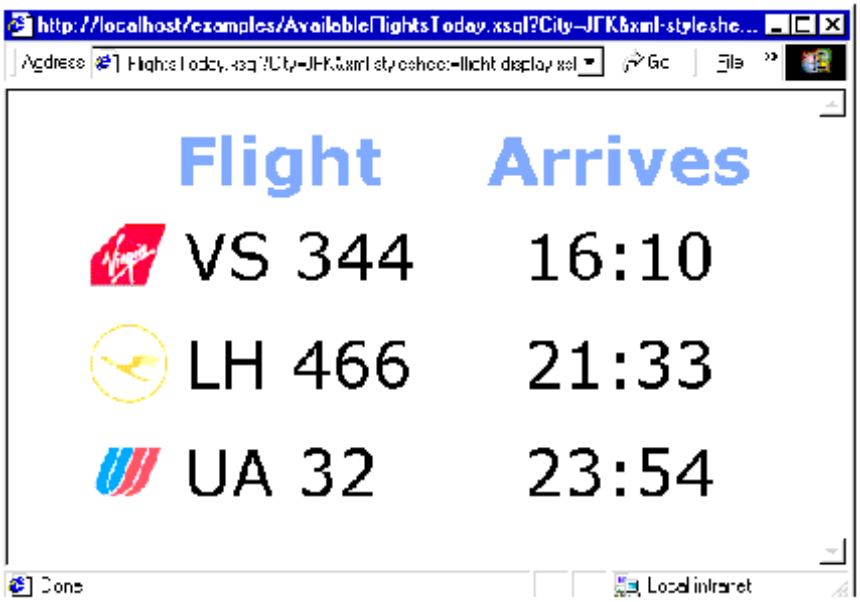
```
Due
      FROM FlightSchedule
     WHERE TRUNC(ExpectedTime) = TRUNC(SYSDATE) AND Arrived = 'N'
       AND Destination = ?   /* The ? is a bind variable being bound */
      ORDER BY ExpectedTime  /* to the value of the City parameter   */
</xsql:query>
```

24.3.4 Transforming XML Datagrams into HTML for Display

To return XML data in HTML instead of an alternative XML format, use an appropriate XSLT stylesheet. For example, rather than producing elements such as `<flight-list>` and `<flight>`, you can write a stylesheet that produces HTML elements such as `<table>`, `<tr>`, and `<td>`.

The result of the dynamically queried data then looks like the HTML page shown in Figure 24-7. Instead of returning raw XML data, the XSQL page leverages server-side XSLT transformation to format the information as HTML for delivery to the browser.

Figure 24-7 Using an XSLT Stylesheet to Render HTML



Similar to the syntax of the `flight-list.xsl` stylesheet, the `flight-display.xsl` stylesheet shown in Example 24-15 looks like a template HTML page. It contains `<xsl:for-each>`, `<xsl:value-of>`, and attribute value templates such as `{DUE}` to plug in the dynamic values from the underlying `<ROWSET>` and `<ROW>` structured XML query results.

> **Note**
>
> The stylesheet produces well-formed HTML. Each opening tag is properly closed (for example, `<td>…</td>`); empty tags use the XML empty element syntax `<br/>` instead of just `<br>`.

Copyright © 1999, 2025, Oracle and/or its affiliates.

You can achieve useful results quickly by combining the power of:

- Parameterized SQL statements to select information from Oracle Database
- Industry-standard XML as a portable, interim data exchange format
- XSLT to transform XML-based datagrams into any XML- or HTML-based format

Example 24-15 Query Results in flight-display.xsl

```
<!-- 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><img align="absmiddle" src="images/{CARRIER}.gif"/></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>
```

24.4 Using XSQL in Java Programs

Class `oracle.xml.xsql.XSQLRequest` lets you use the XSQL page processor in your Java programs.

To use the XSQL Java API, follow these basic steps:

1. Construct an instance of `XSQLRequest`, passing the XSQL page to be processed into the constructor as one of these components:
   - `String` containing a URL to the page
   - `URL` object for the page
   - In-memory `XMLDocument`
2. Invoke one of these methods on the object to process the page:
   - `process()` to write the result to a `PrintWriter` or `OutputStream`
   - `processToXML()` to return the result as an XML Document

Copyright © 1999, 2025, Oracle and/or its affiliates.

To use the built-in XSQL connection manager, which implements JDBC connection pooling based on XSQL configuration file definitions, the XSQL page is all you must pass to the constructor. Optionally, you can pass in a custom implementation for the `XSQLConnectionManagerFactory` interface as well.

The ability to pass the XSQL page as an in-memory `XMLDocument` object means that you can dynamically generate any valid XSQL page for processing. You can then pass the page to the XSQL engine for evaluation.

When processing a page, you may want to perform these additional tasks as part of the request:

- Pass a set of parameters to the request.

  You accomplish this aim 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 by using the `XSQLResquest.setPostedDocument()` method.

Example 24-16 shows how you can process a page by using `XSQLRequest`.

> **See Also**
>
> Using the XSQL Pages Publishing Framework: Advanced Topics to learn more about the XSQL Java API

Example 24-16 XSQLRequestSample Class

```
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);
   // Set up 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));
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
  }
}
```

Related Topics

- Using the XSQL Pages Publishing Framework: Advanced Topics
  An explanation is given of how to use advanced features of the XSQL pages publishing framework.

24.5 XSQL Pages Tips and Techniques

Topics here provide information about using XSQL pages.

24.5.1 XSQL Pages Limitations

Limitations are specified for XSQL pages.

HTTP parameters with multibyte names, such as a parameter whose name is in Kanji, are properly handled when they are inserted into your XSQL page with element <xsql:include-request-params>. An attempt to refer to a parameter with a multibyte name inside the query statement of an <xsql:query> tag returns an empty string for the parameter value.

As a workaround, use a nonmultibyte parameter name. The parameter can still have a multibyte value that can be handled correctly.

24.5.2 Hints for Using the XSQL Servlet

Topics here provide hints for using the XSQL Servlet.

24.5.2.1 Specifying a DTD While Transforming XSQL Output to a WML Document

You can specify a DTD while transforming XSQL output to a Wireless Markup Language (WML) document for a wireless application. The technique is to use a built-in facility of the XSLT stylesheet called `<xsl:output>`. An example illustrates this.

```
<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>
```

The preceding stylesheet produces an XML result that includes this code, where *"your.dtd"* can be any valid absolute or relative URL:

```
<!DOCTYPE xxxx SYSTEM "your.dtd">
```

24.5.2.2 Testing Conditions in XSQL Pages

You can include if-then logic in your XSQL pages.

Example 24-17 shows a technique for executing a query based on a test of a parameter value.

Copyright © 1999, 2025, Oracle and/or its affiliates.

> **See Also**
>
> XSQL Pages Reference to learn about the <xsql:if-param> action

Example 24-17 Conditional Statements in XSQL Pages

```
<xsql:if-param name="security" equals="admin">
  <xsql:query>
      SELECT ....
  </xsql:query>
</xsq:when>
<xsql:if-param name="security" equals="user">
  <xsql:query>
      SELECT ....
  </xsql:query>
</xsql:if-param>
```

24.5.2.3 Passing a Query Result to the WHERE Clause of Another Query

If you have two queries in an XSQL page then you can use the value of a select list item of the first query in the second query by using page parameters.

Example 24-18 Passing Values Among SQL Queries

```
<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 ...
  </xsl:set-param-param>
  <xsql:query bind-params="xxx">
    SELECT col3,col4 FROM table2
    WHERE col3 = ?
  </xsql:query>
</page>
```

24.5.2.4 Handling Multivalued HTML Form Parameters

In some situations, you might need to process multivalued HTML `<form>` parameters that are needed for `<input name="choices" type="checkbox">`. Use the parameter array notation on your parameter name (for example, `choices[]`) to refer to the array of values from the selected check boxes.

Assume that you have a multivalued parameter named `guy`. You can use the array parameter notation in an XSQL page as shown in Example 24-19.

Assume that you request this page is requested with this URL, which contains multiple parameters of the same name to produce a multivalued attribute:

```
http://yourserver.com/page.xsql?guy=Curly&guy=Larry&guy=Moe
```

The page returned looks like this:

```
<page>
  <guy-list>Curly,Larry,Moe</guy-list>
  <quoted-guys>'Curly','Larry','Moe'</quoted-guys>
  <guy>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
    <value>Curly</value>
    <value>Larry</value>
    <value>Moe</value>
  </guy>
</page>
```

You can also use the value of a multivalued page parameter in a SQL statement `WHERE` clause by using the code shown in Example 24-20.

Example 24-19 Handling Multivalued Parameters

```
<page xmlns:xsql="urn:oracle-xsql">
  <xsql:set-page-param name="guy-list" value="{@guy[]}"
                       treat-list-as-array="yes"/>
  <xsql:set-page-param name="quoted-guys" value="{@guy[]}"
                       treat-list-as-array="yes" quote-array-values="yes"/>
  <xsql:include-param name="guy-list"/>
  <xsql:include-param name="quoted-guys"/>
  <xsql:include-param name="guy[]"/>
</page>
```

Example 24-20 Using Multivalued Page Parameters in a SQL Statement

```
<page connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:set-page-param name="quoted-guys" value="{@guy[]}"
                       treat-list-as-array="yes"
                       quote-array-values="yes"/>
  <xsql:query>
    SELECT *
    FROM   sometable
    WHERE  name IN ({@quoted-guys})
  </xsql:query>
</page>
```

24.5.2.5 Invoking PL/SQL Wrapper Procedures to Generate XML Datagrams

The use of PL/SQL wrapper procedures to generate XML datagrams is described.

You cannot set parameter values by binding them in the position of `OUT` variables with <xsql:dml>. Only `IN` parameters are supported for binding. You can create a wrapper procedure, however, that constructs XML elements with the HTTP package. Your XSQL page can then invoke the wrapper procedure with <xsql:include-owa>.

Example 24-21 shows a PL/SQL procedure that accepts two `IN` parameters, multiplies them and puts the value in one `OUT` parameter, then adds them and puts the result in a second `OUT` parameter.

You can write the PL/SQL procedure in Example 24-22 to wrap the procedure in Example 24-21. The `addmultwrapper` procedure accepts the `IN` arguments that the `addmult` procedure preceding expects, and then encodes the `OUT` values as an XML datagram that you print to the Open Web Analytics (OWA) page buffer.

The XSQL page shown in Example 24-23 constructs an XML document by including a call to the PL/SQL wrapper procedure.

You can invoke `addmult.xsql` by entering a URL in a browser:

```
http://yourserver.com/addmult.xsql?arg1=30&arg2=45
```

The XML datagram returned by the servlet reflects the `OUT` values:

Copyright © 1999, 2025, Oracle and/or its affiliates.

```
<page>
  <addmult><sum>75</sum><product>1350</product></addmult>
</page>
```

Example 24-21 addmult PL/SQL 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;
```

Example 24-22 addmultwrapper PL/SQL Procedure

```
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;
```

Example 24-23 addmult.xsql

```
<page connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:include-owa bind-params="arg1 arg2">
    BEGIN addmultwrapper(?,?); END;
  </xsql:include-owa>
</page>
```

24.5.2.6 Accessing Contents of Posted XML

The XSQL page processor can access the contents of posted XML. Any XML document can be posted and handled by the feature that XSQL supports.

For example, an XSQL page can access the contents of an inbound SOAP message by using the `xpath="`*`XpathExpression`*`"` attribute in the <xsql:set-page-param> action. Alternatively, custom action handlers can gain direct access to the SOAP message body by invoking `getPageRequest().getPostedDocument()`. To create the SOAP response body to return to the client, use an XSLT stylesheet or a custom serializer implementation to write the XML response in an appropriate SOAP-encoded format.

> **See Also**
>
> The Airport SOAP demo for an example of using an XSQL page to implement a SOAP-based web service

Copyright © 1999, 2025, Oracle and/or its affiliates.

24.5.2.7 Changing Database Connections Dynamically

You can choose database connections dynamically when invoking an XSQL page. For example, you might want to switch between a test database and a production database. You can achieve this goal by including an XSQL parameter in the `connection` attribute of the XSQL page.

Define an attribute of the same name to serve as the default value for the connection name.

Assume that in your XSQL configuration file you define connections for database `testdb` and `proddb`. You then write an XSQL page with this <xsql:query> element:

```
<xsql:query conn="testdb" connection="{@conn}" xmlns:xsql="urn:oracle-xsql">
  ...
</xsql:query>
```

If you request this page without any parameters, then the value of the `conn` parameter is `testdb`, so the page uses the connection named `testdb` defined in the XSQL configuration file. If you request the page with `conn=proddb`, then the page uses the connection named `proddb` instead.

24.5.2.8 Retrieving the Name of the Current XSQL Page

An XSQL page can access its own name in a generic way at run time to construct links to the current page.

You can use a helper method like the one shown in Example 24-24 to retrieve the name of the page inside a custom action handler.

Example 24-24 Getting the Name of the Current XSQL Page

```
private String curPageName(XSQLPageRequest req) {
  String thisPage = req.getSourceDocumentURI();;
  int pos = thisPage.lastIndexOf('/');
  if (pos >=0) thisPage = thisPage.substring(pos+1);
  pos = thisPage.indexOf('?');
  if (pos >=0) thisPage = thisPage.substring(0,pos-1);
  return thisPage;
}
```

24.5.3 Resolving Common XSQL Connection Errors

Topics here include receiving unable-to-connect and no-posted-document errors.

24.5.3.1 Receiving "Unable to Connect" Errors

Reasons are given for receiving errors saying that you cannot connect.

Suppose you are unable to connect to a database and you see errors similar to these when running the `helloworld.xsql` sample program:

```
Oracle XSQL Servlet Page Processor
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))))
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

The preceding errors indicate that the XSQL servlet is attempting the JDBC connection based on the <connectiondef> information for the connection named demo, assuming you did not modify the helloworld.xsql demo page.

By default the XSQLConfig.xml file comes with the entry for the demo connection that looks like this (use the correct password):

```
<connection name="demo">
  <username>scott</username>
  <password>password</password>
  <dburl>jdbc:oracle:thin:@localhost:1521:ORCL</dburl>
  <driver>oracle.jdbc.driver.OracleDriver</driver>
</connection>
```

The error is probably due to one of these reasons:

- Your database is not on the localhost machine.
- Your database SID is not ORCL.
- Your TNS Listener Port is not 1521.

24.5.3.2 Receiving "No Posted Document to Process" When Using HTTP POST

If you try to post XML information to an XSQL page for processing using HTTP GET instead of HTTP POST, then there is no posted document, and you get the “No posted document to process” error.

XML information posted to an XSQL page for processing must be sent by HTTP POST. This transfer can be effected by an HTML form or an XML document sent by HTTP POST.

24.5.4 Security Considerations for XSQL Pages

Best practices are covered for managing security in the XSQL servlet.

24.5.4.1 Installing Your XSQL Configuration File in a Safe Directory

The XSQLConfig.xml configuration file contains sensitive database user name and password information. This file must not reside in any directory that maps to a virtual path of your web server, nor in any of its subdirectories.

The only required permissions for the configuration file are read permission granted to the UNIX account that owns the servlet engine. Failure to follow this recommendation could mean that a user of your site could browse the contents of your configuration file, thereby getting the passwords to database accounts.

24.5.4.2 Disabling Default Client Stylesheet Overrides

By default, the XSQL page processor lets you supply a stylesheet in a page request by passing a value for parameter xml-stylesheet. If you want the stylesheet referenced by the server-side XSQL page to be the only legal stylesheet, then include attribute allow-client-style="no" on the document element of your page.

You can also globally change the default setting in the XSQLConfig.xml file to disallow client stylesheet overrides. If you take either approach, then the only pages that allow client stylesheet overrides are those that include the allow-client-style="yes" attribute on their document element.

Copyright © 1999, 2025, Oracle and/or its affiliates.

24.5.4.3 Protecting Against the Misuse of Substitution Parameters

Some precautions are described that help you avoid misuse of substitution variables.

Any product that supports the use of lexical substitution variables in a SQL query can cause a developer problems. Any time you deploy an XSQL page that allows part of all of a SQL statement to be substituted by a lexical parameter, you must ensure that you have taken appropriate precautions against misuse.

For example, one of the demonstrations that comes with XSQL Pages is the Adhoc Query Demo. It shows how you can supply the entire SQL statement of an <xsql:query> action handler as a parameter. This technique is a powerful and beneficial tool when in the right hands, but if you deploy a similar page to your production system, then the user can execute any query that the database security privileges for the connection associated with the page allows. For example, the Adhoc Query Demo is set up to use a connection that maps to the `scott` account, so a user can query any data that `scott` would be allowed to query from SQL*Plus.

You can use these techniques to ensure that your pages are not abused:

- Ensure the database user account associated with the page has only the privileges for reading the tables and views you want your users to see.
- Use true bind variables instead of lexical bind variables when substituting single values in a `SELECT` statement. If you must parameterize syntactic parts of your SQL statement, then lexical parameters are the only way to proceed. Otherwise, use true bind variables so that any attempt to pass an invalid value generates an error instead of producing an unexpected result.

Copyright © 1999, 2025, Oracle and/or its affiliates.

25

Using the XSQL Pages Publishing Framework: Advanced Topics

An explanation is given of how to use advanced features of the XSQL pages publishing framework.

Related Topics

- Using the XSQL Pages Publishing Framework
  An explanation is given of how to use the basic features of the XSQL pages publishing framework.

> **See Also**
>
> Using the XSQL Pages Publishing Framework for information about basic features

25.1 Customizing the XSQL Configuration File Name

By default, the XSQL pages framework expects the configuration file to be named `XSQLConfig.xml`. When moving between development, test, and production environments, you can switch among different versions of a configuration file. To override the name of the configuration file read by the XSQL page processor, set system property `xsql.config`.

The simplest technique is to specify a Java Virtual Machine (JVM) command-line flag such as `-Dxsql.config=`*`MyConfigFile.xml`* by defining a servlet initialization parameter named `xsql.config`. Add an `<init-param>` element to your `web.xml` file as part of the `<servlet>` tag that defines the XSQL Servlet:

```
<servlet>
  <servlet-name>XSQL</servlet-name>
  <servlet-class>oracle.xml.xsql.XSQLServlet</servlet-class>
  <init-param>
    <param-name>xsql.config</param-name>
    <param-value>MyConfigFile.xml</param-value>
    <description>
       Please Use MyConfigFile.xml instead of XSQLConfig.xml
    </description>
  </init-param>
</servlet>
```

The servlet initialization parameter is applicable only to the servlet-based use of the XSQL engine. When using the `XSQLCommandLine` or `XSQLRequest` programmatic interfaces, use the `System` parameter instead.

Copyright © 1999, 2025, Oracle and/or its affiliates.

> **Note**
>
> The configuration file is always read from the `CLASSPATH`. For example, if you specify a custom configuration parameter file named `MyConfigFile.xml`, then the XSQL processor attempts to read the XML file as a resource from the `CLASSPATH`. In a servlet environment like Java Platform, Enterprise Edition (Java EE), you must place your `MyConfigFile.xml` in the `.\WEB-INF\classes` directory (or another top-level directory on the `CLASSPATH`). If both the servlet initialization parameter and the `System` parameter are provided, then the servlet initialization parameter value is used.

25.2 Controlling How Stylesheets Are Processed

Topics here include an overview of client stylesheets, controlling content type, assigning stylesheets dynamically, processing stylesheets in a client, and providing multiple stylesheets.

25.2.1 Overriding Client Stylesheets

If the current XSQL page being requested allows it, you can supply an Extensible Stylesheet Language Transformation (XSLT) stylesheet URL in the request. This technique lets you override the default stylesheet or apply a stylesheet where none is 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 that uses the HTTP protocol scheme, provided it references a trusted host as defined in the XSQL configuration file.
- The literal value `none`. Setting `xml-stylesheet=none` is useful during development to temporarily "short-circuit" the XSLT stylesheet processing to determine what XML datagram your stylesheet is seeing. Use this technique to determine why a stylesheet is not producing expected results.

You can allow client override of stylesheets for an XSQL page in these ways:

- Setting the `allow-client-style` configuration parameter to `no` in the XSQL configuration file
- 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 XSQL 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.

25.2.2 Controlling the Content Type of the Returned Document

Setting the content type of the data you serve lets a requesting client correctly interpret the data you return. If your stylesheet uses an `<xsl:output>` element then the XSQL processor

Copyright © 1999, 2025, Oracle and/or its affiliates.

infers the media type and the encoding of the returned document from the `media-type` and `encoding` attributes of `<xsl:output>`.

The stylesheet in Example 25-1 uses the `media-type="application/vnd.ms-excel"` attribute on `<xsl:output>`. This instruction transforms the results of an XSQL page containing a standard query of the `hr.employees` table into Microsoft Excel format.

The following XSQL page uses the stylesheet in Example 25-1:

```
<?xml version="1.0"?>
<?xml-stylesheet href="empToExcel.xsl" type="text/xsl"?>
<xsql:query connection="hr" xmlns:xsql="urn:oracle-xsql">
  SELECT   employee_id, email, salary
  FROM     employees
  ORDER BY salary DESC
</xsql:query>
```

Example 25-1 empToExcel.xsl

```
<?xml version="1.0"?>
<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>Id</th><th>Email</th><th>Salary</th></tr>
       <xsl:for-each select="ROWSET/ROW">
         <tr>
           <td><xsl:value-of select="EMPLOYEE_ID"/></td>
           <td><xsl:value-of select="EMAIL"/></td>
           <td><xsl:value-of select="SALARY"/></td>
         </tr>
       </xsl:for-each>
     </table>
   </html>
  </xsl:template>
</xsl:stylesheet>
```

25.2.3 Assigning the Stylesheet Dynamically

If you include an `<?xml-stylesheet?>` instruction at the top of your `.xsql` file, then the XSQL page processor considers it for use in transforming the resulting XML datagram.

Consider the `emp_test.xsql` page shown in Example 25-2.

The page in Example 25-2 uses the `emp.xsl` stylesheet to transform the results of the `employees` query in the server tier before returning the response to the requester. The processor accesses the stylesheet by the URL provided in the `href` pseudo-attribute on the `<?xml-stylesheet?>` processing instruction.

For example, to change XSLT stylesheets dynamically based on arguments passed to the XSQL servlet, you can use a lexical parameter in the `href` attribute of your `xml-stylesheet` processing instruction, as shown in this sample instruction:

```
<?xml-stylesheet type="text/xsl" href="{@filename}.xsl"?>
```

You can then pass the value of the `filename` parameter as part of the URL request to XSQL servlet.

Copyright © 1999, 2025, Oracle and/or its affiliates.

You can also use the <xsql:set-page-param> element in an XSQL page to set the value of the parameter based on a SQL query. For example, the XSQL page in Example 25-3 selects the name of the stylesheet to use from a table by assigning the value of a page-private parameter.

Example 25-2 emp_test.xsql

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="emp.xsl"?>
<page connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:query>
    SELECT   *
    FROM     employees
    ORDER BY salary DESC
  </xsql:query>
</page>
```

Example 25-3 emp_test_dynamic.xsql

```
<?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     employees
    ORDER BY salary DESC
  </xsql:query>
</page>
```

25.2.4 Processing XSLT Stylesheets in the Client

How to process XSLT stylesheets in the client is described.

Some browsers support processing XSLT stylesheets in the client. These browsers recognize the stylesheet to be processed for an XML document by using an `<?xml-stylesheet?>` processing instruction. 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 pages processor performs XSLT transformations in the server. By adding `client="yes"` to your `<?xml-stylesheet?>` processing instruction in your XSQL page, however, you can defer XSLT processing to the client. The processor serves the XML datagram "raw" with the current `<?xml-stylesheet?>` element at the top of the document.

25.2.5 Providing Multiple Stylesheets

You can include multiple `<?xml-stylesheet?>` processing instructions at the top of an XSQL page.

The instructions can contain an optional `media` pseudo-attribute. If specified, the processor case-insensitively compares the value of the `media` pseudo-attribute with the value of the User-Agent string in the HTTP header. If the value of the `media` pseudo-attribute matches part of the User-Agent string, then the processor selects the current `<?xml-stylesheet?>` instruction for use. Otherwise, the processor ignores the instruction and continues looking. The processor

Copyright © 1999, 2025, Oracle and/or its affiliates.

uses the first matching processing instruction in document order. An instruction *without* a `media` pseudo-attribute matches all user agents.

Processing Instructions"?> shows multiple processing instructions at the top of an XSQL file. The processor uses `doyouxml-lynx.xsl` for Lynx browsers, `doyouxml-ie.xsl` for Internet Explorer 5.0 or 5.5 browsers, and `doyouxml.xsl` for all others.

"?> summarizes the supported pseudo-attributes allowed on the `<?xml-stylesheet?>` processing instruction.

Table 25-1 Pseudo-Attributes for <?xml-stylesheet ?>

Attribute Name	Description
`type = "string"`	Indicates the Multipurpose Internet Mail Extensions (MIME) type of the associated stylesheet. For XSLT stylesheets, this attribute must be set to the string `text/xsl`. This attribute may be present *or* absent when using the `serializer` attribute, depending on whether an XSLT stylesheet must execute before invoking the serializer, or not.
`href = "URL"`	Indicates the relative or absolute URL to the XSLT stylesheet to be used. If an absolute URL is supplied that uses the `http` protocol scheme, the IP address of the resource must be a trusted host listed in the XSQL configuration file (by default, named `XSQLConfig.xml`).
`media = "string"`	Performs a case-*insensitive* match on the `User-Agent` string from the HTTP header sent by the requesting device. This attribute is optional. The current `<?xml-stylesheet?>` processing instruction is used only if the `User-Agent` string contains the value of the `media` attribute; otherwise it is ignored.
`client = "boolean"`	Defers the processing of the associated XSLT stylesheet to the client if set to `yes`. The raw XML datagram is sent to the client with the current `<?xml-stylesheet?>` instruction at the top of the document. The default if not specified is to perform the transformation in the server.
`serializer = "string"`	By default, the XSQL page processor uses: • XML Document Object Model (DOM) serializer if no XSLT stylesheet is used • XSLT processor serializer if an XSLT stylesheet is used Specifying this pseudo-attribute indicates that a custom serializer implementation must be used instead. Valid values are either the name of a custom serializer defined in the `<serializerdefs>` section of the XSQL configuration file or the string `java:fully.qualified.Classname`. If both an XSLT stylesheet and the serializer attribute are present, then the processor performs the XSLT transformation first, then invokes the custom serializer to render the final result to the `OutputStream` or `PrintWriter`.

Example 25-4 Multiple <?xml-stylesheet ?> Processing Instructions

```
<?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">
```

25.3 Working with Array-Valued Parameters

Topics here include using array values for parameters, including page or session parameters and parameters in SQL or PL/SQL code

Copyright © 1999, 2025, Oracle and/or its affiliates.

25.3.1 Supplying Values for Array-Valued Parameters

Request parameters, session parameters, and page-private parameters can have arrays of strings as values. To treat to the value of a parameter as an array, add two empty square brackets to the end of its name.

For example, if an HTML form is posted with four occurrences of a input control named `productid`, then use the notation `productid[]` to refer to the array-valued `productid` parameter. If you refer to an array-valued parameter without using the array-brackets notation, then the XSQL processor uses the value of the first array entry.

> **Note**
>
> The XSQL processor does not support use of numbers inside the array brackets. That is, you can refer to `productid` or `productid[]`, but not `productid[2]`.

Suppose that you refer to an array-valued parameter as a lexical substitution parameter inside an action handler attribute value or inside the content of an action handler element. The XSQL page processor converts its value to a comma-delimited list of non-null and nonempty strings in the order that they exist in the array. Example 25-5 shows an XSQL page with an array-valued parameter.

You can invoke the XSQL command-line utility to supply multiple values for the `productid` parameter in `Page.xsql`:

```
xsql Page.xsql productid=111 productid=222 productid=333 productid=444
```

The preceding command sets the `productid[]` array-valued parameter to the value {"111","222","333","444"}. The XSQL page processor replaces the `{@productid[]}` expression in the query with the string "111,222,333,444".

You can also pass multivalued parameters programmatically through the `XSQLRequest` application programming interface (API), which accepts a `java.util.Dictionary` of named parameters. You can use a `Hashtable` and invoke its `put(name,value)` method to add `String`-valued parameters to the request. To add multivalued parameters, put a value of type `String[]` instead of type `String`.

> **Note**
>
> Only request parameters, page-private parameters, and session parameters can use string arrays. The <xsql:set-stylesheet-param> and <xsql:set-cookie> actions support only working with parameters as simple string values. To refer to a multivalued parameter in your XSLT stylesheet, use <xsql:include-param> to include the multivalued parameter into your XSQL datapage, then use an XPath expression in the stylesheet to refer to the values from the datapage.

Example 25-5 Using an Array-Valued Parameter in an XSQL Page

```
<page xmlns:xsql="urn:oracle-xsql">
  <xsql:query>
    SELECT description
    FROM product
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
    WHERE productid in ( {@productid[]} )  /* Using lexical parameter */
  </xsql:query>
</page>
```

25.3.2 Setting Array-Valued Page or Session Parameters from Strings

You can set the value of a page-private parameter or a session parameter from strings.

You can set the value of a page-private parameter to a string-array value by using array brackets notation on the name:

```
<!-- param name contains array brackets -->
<xsql:set-page-param name="names[]" value="Tom Jane Joe"/>
```

You set the value similarly for session parameters, as shown in this example:

```
<xsql:set-session-param name="dates[]" value="12-APR-1962 15-JUL-1968"/>
```

By default, when the name of the parameter uses array brackets, the XSQL processor treats the value as a space-or-comma-delimited list and tokenizes it.

The resulting string array value contains these separate tokens. In the preceding examples, parameter `names[]` is the string array {"`Tom`", "`Jane`", "`Joe`"} and parameter `dates[]` is the string array {"`12-APR-1962`", "`15-JUL-1968`"}.

To handle strings that contain spaces, the tokenization algorithm first checks the string for the presence of commas. If at least one comma is found in the string, then commas are used as the token delimiter. For example, this action sets the value of the `names[]` parameter to the string array {"`Tom Jones`", "`Jane York`"}:

```
<!-- param name contains array brackets -->
<xsql:set-page-param name="names[]" value="Tom Jones,Jane York"/>
```

By default, when you set a parameter whose name does not end with the array-brackets, then the string-tokenization does not occur. Thus, this action sets the parameter `names` to the literal string "`Tom Jones,Jane York`":

```
<!-- param name does NOT contain array brackets -->
<xsql:set-page-param name="names" value="Tom Jones,Jane York"/>
```

You can force the string to be tokenized by including the `treat-list-as-array="yes"` attribute on the <xsql:set-page-param> or <xsql:set-session-param> actions. When this attribute is set, the XSQL processor assigns a comma-delimited string of the tokenized values to the parameter. For example, this action sets the `names` parameter to the literal string "`Tom,Jane,Joe`":

```
<!-- param name does NOT contain array brackets -->
<xsql:set-page-param name="names" value="Tom Jane Joe"
                     treat-list-as-array="yes"/>
```

When you are setting the value of a simple string-valued parameter and you are tokenizing the value with `treat-list-as-array="yes"`, you can include the `quote-array-values="yes"` attribute to surround the comma-delimited values with single quotation marks. Thus, this action assigns the literal string value "`'Tom Jones','Jane York','Jimmy'`" to the `names` parameter:

```
<!--  param name does NOT contain array brackets -->
<xsql:set-page-param name="names" value="Tom Jones,Jane York,Jimmy"
                     treat-list-as-array="yes"
                     quote-array-values="yes"/>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

25.3.3 Binding Array-Valued Parameters in SQL and PL/SQL Statements

Where string-valued scalar bind variables are supported in an XSQL page, you can also bind array-valued parameters. Use the array parameter name, for example, `myparam[]`, in the list of parameter names that you supply for attribute `bind-params`. This technique lets you process array-valued parameters in SQL statements and PL/SQL procedures.

The XSQL processor binds array-valued parameters as a nested table object type named `XSQL_TABLE_OF_VARCHAR`. You must create this type in your current schema with this DDL statement:

```
CREATE TYPE xsql_table_of_varchar AS TABLE OF VARCHAR2(2000);
```

Although the type must have the name `xsql_table_of_varchar`, you can change the dimension of the `VARCHAR2` string, if necessary. You must make the dimension long enough for any string value you expect to handle in your array-valued string parameters.

Consider the PL/SQL function shown in Example 25-6.

The XSQL page in Example 25-7 shows how to bind two array-valued parameters in a SQL statement that uses `testTableFunction`.

Executing the XSQL page in Example 25-7 generates this datagram:

```
<page someNames="aa,bb,cc" someValues="11,22,33">
  <ROWSET>
    <ROW num="1">
      <EXAMPLE>aa=11:bb=22:cc=33</EXAMPLE>
    </ROW>
  </ROWSET>
</page>
```

This technique shows that the XSQL processor bound the array-valued `someNames[]` and `someValues[]` parameters as table collection types. It iterated over the values and concatenated them to produce the "`aa=11:bb=22:cc=33`" string value as the return value of the PL/SQL function.

You can mix any number of regular parameters and array-valued parameters in your bind-params string. Use the array-bracket notation for the parameters to be bound as arrays.

> **Note**
>
> If you run the page in Example 25-7 but you have not created the `XSQL_TABLE_OF_VARCHAR` type as showd earlier, then you receive an error such as:

```
<page someNames="aa,bb,cc" someValues="11,22,33">
  <xsql-error code="17074" action="xsql:query">
    <statement>
     select testTableFunction(?,?) as example from dual
    </statement>
    <message>
      invalid name pattern: SCOTT.XSQL_TABLE_OF_VARCHAR
    </message>
  </xsql-error>
</page>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

Because the XSQL processor binds array parameters as nested table collection types, you can use the `TABLE()` operator with the `CAST()` operator in SQL to treat the nested table bind variable value as a table of values. You can then query this table. This technique is especially useful in subqueries. The page in Example 25-8 uses an array-valued parameter containing employee IDs to restrict the rows queried from `hr.employees`.

The XSQL page in Example 25-8 generates a datagram such as:

```
<page>
  <ROWSET>
    <ROW num="1">
      <NAME>Alana Walsh</NAME>
      <SALARY>3100</SALARY>
    </ROW>
    <ROW num="2">
      <NAME>Kevin Feeny</NAME>
      <SALARY>3000</SALARY>
    </ROW>
  </ROWSET>
</page>
```

Example 25-7 and Example 25-8 show how to use `bind-params` with <xsql:query>, but these techniques work for <xsql:dml>, <xsql:include-owa>, <xsql:ref-cursor-function>, and other actions that accept SQL or PL/SQL statements.

PL/SQL index-by tables work with the OCI JDBC driver but not the JDBC thin driver. By using the nested table collection type `XSQL_TABLE_OF_VARCHAR`, you can use array-valued parameters with either driver. In this way you avoid losing the programming flexibility of working with array values in PL/SQL.

Example 25-6 testTableFunction

```
FUNCTION testTableFunction(p_name  XSQL_TABLE_OF_VARCHAR,
                           p_value XSQL_TABLE_OF_VARCHAR)
RETURN VARCHAR2 IS
  lv_ret     VARCHAR2(4000);
  lv_numElts INTEGER;
BEGIN
  IF p_name IS NOT NULL THEN
    lv_numElts := p_name.COUNT;
    FOR j IN 1..lv_numElts LOOP
      IF (j > 1) THEN
        lv_ret := lv_ret||':';
      END IF;
      lv_ret := lv_ret||p_name(j)||'='||p_value(j);
    END LOOP;
  END IF;
  RETURN lv_ret;
END;
```

Example 25-7 XSQL Page with Array-Valued Parameters

```
<page xmlns:xsql="urn:oracle-xsql" connection="demo"
      someNames="aa,bb,cc" someValues="11,22,33">
  <xsql:query bind-params="someNames[] someValues[]">
    SELECT testTableFunction(?,?) AS example
    FROM dual
  </xsql:query>
</page>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

Example 25-8 Using an Array-Valued Parameter to Restrict Rows

```
<page xmlns:xsql="urn:oracle-xsql" connection="hr">
  <xsql:set-page-param name="someEmployees[]" value="196,197"/>
  <xsql:query bind-params="someEmployees[]">
    SELECT first_name||' '||last_name AS name, salary
    FROM employees
    WHERE employee_id IN (
        SELECT * FROM TABLE(CAST( ? AS xsql_table_of_varchar))
     )
   </xsql:query>
</page>
```

25.4 Setting Error Parameters on Built-In Actions

You can set a page-private parameter on a built-in XSQL action when the action reports a nonfatal error.

The XSQL page processor determines whether an action encountered a nonfatal error during its execution. For example, an attempt to insert a row or invoke a stored procedure can fail with a database exception that gets included in your XSQL data page as an `<xsql-error>` element.

Use attribute `error-param` on the action to set a page-private parameter on a built-in XSQL action when the action reports a nonfatal error. For example, to set parameter `dml-error` when the statement inside action <xsql:dml> encounters a database error, you can use the technique shown in Example 25-9.

If the execution of action `<xsql:dml>` encounters an error then the XSQL processor sets the page-private parameter `dml-error` to the string `"Error"`. If the execution is successful then the XSQL processor does not assign a value to the error parameter. In Example 25-9, if the page-private parameter `dml-error` already exists then it retains its current value. If it does not exist then it continues not to exist.

Example 25-9 Setting an Error Parameter

```
<xsql:dml error-param="dml-error" bind-params="val">
  INSERT INTO yourtable(somecol)
    VALUES(?)
</xsql:dml>
```

25.4.1 Using Conditional Logic with Error Parameters

How to get conditional behavior in your XSQL page template is described.

By using the error parameter in combination with <xsql:if-param>, you can achieve conditional behavior in your XSQL page template. For example, assume that your connection definition sets the `AUTOCOMMIT` flag to `false` on the connection named `demo` in the XSQL configuration file. The XSQL page shown in Example 25-10 shows how you might roll back the changes made by a previous action if a subsequent action encounters an error.

If you have written custom action handlers, and if your custom actions invoke `reportMissingAttribute()`, `reportError()`, or `reportErrorIncludingStatement()` to report nonfatal action errors, then they automatically pick up this feature as well.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Example 25-10 Achieving Conditional Behavior with an Error Parameter

```
<!-- NOTE: Connection "demo" must not set to autocommit! -->
<page connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:dml error-param="dml-error" bind-params="val">
    INSERT INTO yourtable(somecol)
      VALUES(?)
  </xsql:dml>
  <!-- This second statement will commit if it succeeds -->
  <xsql:dml commit="yes" error-param="dml-error" bind-params="val2">
    INSERT INTO anothertable(anothercol)
      VALUES(?)
  </xsql:dml>
  <xsql:if-param name="dml-error" exists="yes">
    <xsql:dml>
      ROLLBACK
    </xsql:dml>
  </xsql:if-param>
</page>
```

25.4.2 Formatting XSQL Action Handler Errors

Errors raised by the processing of XSQL action elements are reported as XML elements in a uniform way. This fact enables XSLT stylesheets to detect their presence and optionally format them for presentation.

The action element in error is replaced in the page by this element:

```
<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

Example 25-11 shows an XSLT stylesheet that uses this information to display error information on the screen.

Example 25-11 XSLT Stylesheet

```
<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>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

25.5 Including XMLType Query Results in XSQL Pages

Oracle Database supports `XMLType` for storing and querying XML-based database content.

You can exploit database XML features to produce XML data for inclusion in your XSQL pages by using one of these techniques:

- <xsql:query> handles any query including columns of type `XMLType`, but it handles XML markup in `CLOB` and `VARCHAR2` columns as literal text.
- <xsql:include-xml> parses and includes a single `CLOB` or string-based XML document retrieved from a query.

One difference between the preceding approaches is that `<xsql:include-xml>` parses the literal XML appearing in a CLOB or string value as needed to turn it into a tree of elements and attributes. In contrast, `<xsql:query>` leaves XML markup in `CLOB` or string-valued columns as literal text.

Another difference is that while `<xsql:query>` can handle query results of any number of columns and rows, `<xsql:include-xml>` works on a single column of a single row. Accordingly, when using `<xsql:include-xml>`, the `SELECT` statement inside it returns a single row containing a single column. The column can either be a `CLOB` or a `VARCHAR2` value containing a well-formed XML document. The XSQL engine parses the XML document and includes it in your XSQL page.

Example 25-12 uses nested `XmlAgg()` functions to aggregate the results of a dynamically-constructed XML document containing departments and nested employees. The functions aggregate the document into a single "result" document wrapped in a `<DepartmentList>` element.

In another example, suppose you have many `<Movie>` XML documents stored in a table of `XMLType` called `movies`. Each document might look like the one shown in Example 25-13.

You can use the built-in XPath query features to extract an aggregate list of all cast members who have received Oscar awards from any movie in the database. Example 25-14 shows a sample query.

To include this query result of `XMLType` in your XSQL page, paste the query inside an `<xsql:query>` element. Make sure you include an alias for the query expression, as shown in Example 25-15.

You can use the combination of `XmlElement()` and `XmlAgg()` to make the database aggregate all of the XML fragments identified by the query into single, well-formed XML document. The functions work to produce a well-formed result like this:

```
<AwardedActors>
  <Actor>...</Actor>
  <Actress>...</Actress>
</AwardedActors>
```

You can use the standard XSQL bind variable capabilities in the middle of an XPath expression if you concatenate the bind variable into the expression. For example, to parameterize the value `Oscar` into a parameter named `award-from`, you can use an XSQL page like the one shown in Example 25-16.

Example 25-12 Aggregating a Dynamically-Constructed XML Document

```
<xsql:query connection="hr" xmlns:xsql="urn:oracle-xsql">
  SELECT XmlElement("DepartmentList",
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
            XmlAgg(
              XmlElement("Department",
                XmlAttributes(department_id AS "Id"),
                XmlForest(department_name AS "Name"),
                (SELECT XmlElement("Employees",
                          XmlAgg(
                            XmlElement("Employee",
                              XmlAttributes(employee_id AS "Id"),
                              XmlForest(first_name||' '||last_name AS "Name",
                                        salary   AS "Salary",
                                        job_id   AS "Job")
                            )
                          )
                        )
                 FROM employees e
                 WHERE e.department_id = d.department_id
                )
              )
            )
          ) AS result
  FROM departments d
  ORDER BY department_name
</xsql:query>
```

Example 25-13 Movie XML Document

```
<Movie Title="The Talented Mr.Ripley" RunningTime="139" Rating="R">
  <Director>
    <First>Anthony</First>
    <Last>Minghella</Last>
  </Director>
  <Cast>
    <Actor Role="Tom Ripley">
      <First>Matt</First>
      <Last>Damon</Last>
    </Actor>
    <Actress Role="Marge Sherwood">
      <First>Gwyneth</First>
      <Last>Paltrow</Last>
    </Actress>
    <Actor Role="Dickie Greenleaf">
      <First>Jude</First>
      <Last>Law</Last>
      <Award From="BAFTA" Category="Best Supporting Actor"/>
    </Actor>
  </Cast>
</Movie>
```

Example 25-14 Using XPath to Extract an Aggregate List

```
SELECT XMLELEMENT("AwardedActors",
           XMLAGG(EXTRACT(VALUE(m),
                  '/Movie/Cast/*[Award[@From="Oscar"]]')))
FROM movies m
```

Example 25-15 Including an XMLType Query Result

```
<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
  SELECT XMLELEMENT("AwardedActors",
           XMLAGG(EXTRACT(VALUE(m),
                  '/Movie/Cast/*[Award[@From="Oscar"]]'))) AS result
  FROM movies m
</xsql:query>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

Example 25-16 Using XSQL Bind Variables in an XPath Expression

```
<xsql:query connection="orcl92" xmlns:xsql="urn:oracle-xsql"
            award-from="Oscar"  bind-params="award-from">
  /* Using a bind variable in an XPath expression */
  SELECT XMLELEMENT("AwardedActors",
           XMLAGG(EXTRACT(VALUE(m),
                  '/Movie/Cast/*[Award[@From="'|| ? ||'"]]'))) AS result
  FROM movies m
</xsql:query>
```

25.6 Handling Posted XML Content

In addition to simplifying the assembly and transformation of XML content, the XSQL pages framework helps you handle posted XML content.

Built-in actions provide these advantages:

- Simplify the handling of posted data from both XML document and HTML forms
- Enable data to be posted directly into a database table by using XSU

XSU can perform database inserts, updates, and deletes based on the content of an XML document in canonical form for a target table or view. For a specified table, the canonical XML form of its data is given by one row of XML output from a `SELECT *` query. When given an XML document in this form, XSU can automate the DML operation.

By combining XSU with XSLT, you can transform XML in any format into the canonical format expected by a given table. XSU can then perform DML on the resulting canonical XML.

The following built-in XSQL actions make it possible for you to exploit this capability from within your XSQL pages:

- <xsql:insert-request>

  Insert the optionally transformed XML document that was posted in the request into a table.

- <xsql:update-request>

  Update the optionally transformed XML document that was posted in the request into a table or view.

- <xsql:delete-request>

  Delete the optionally transformed XML document that was posted in the request from a table or view.

- <xsql:insert-param>

  Insert the optionally transformed XML document that was posted as the value of a request parameter into a table or view.

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 can 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.

25.6.1 Understanding XML Posting Options

An overview is provided of XML posting options.

Copyright © 1999, 2025, Oracle and/or its affiliates.

The XSQL pages framework can handle posted data in these scenarios:

- A client program sends an HTTP `POST` message that targets an XSQL page. The request body contains an XML document; the HTTP header reports a `ContentType` of "`text/xml`".

  In this case, `<xsql:insert-request>`, `<xsql:update-request>`, or `<xsql:delete-request>` can insert, update, or delete the content of the posted XML in the target table. If you transform the posted XML with XSLT, then the posted document is the source for the transformation.

- A client program sends 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 to insert the content of the posted XML parameter value in the target table. If you transform the posted XML document with XSLT, then the XML document in the parameter value is the source document for this transformation.

- A browser submits an HTML form with `method="POST"` whose action targets an XSQL page. The request body of the HTTP `POST` message contains an encoded version of the form fields and values with a `ContentType` of "`application/x-www-form-urlencoded`".

  In this case, the request does not contain an XML document, but an encoded version of the form parameters. To make all three of these cases uniform, however, the XSQL page processor materializes on demand an XML document from the form parameters, session variables, and cookies contained in the request. The XSLT processor transforms this dynamically-materialized XML document into canonical form for DML by using `<xsql:insert>`, `<xsql:update-request>`, or `<xsql:delete-request>`.

When working with posted HTML forms, the dynamically materialized XML document has the form shown in Example 25-17.

If multiple parameters are posted with the same name, then the XSQL processor automatically creates multiple `<row>` elements to make subsequent processing easier. Assume that a request posts or includes these parameters and values:

- `id = 101`
- `name = Steve`
- `id = 102`
- `name = Sita`
- `operation = update`

The XSQL page processor creates a set of parameters as follows:

```
<request>
  <parameters>
    <row>
      <id>101</id>
      <name>Steve</name>
    </row>
    <row>
      <id>102</id>
      <name>Sita</name>
    </row>
    <operation>update</operation>
  </parameters>
  ...
</request>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

You must provide an XSLT stylesheet that transforms this materialized XML document containing the request parameters into canonical format for your target table. Thus, you can build an XSQL page:

```
<!--
 | 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. In the browser you see the "raw" XML for the materialized XML request document, which you can save and use to develop the XSL transformation.

Example 25-17 XML Document Generated from HTML 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>
```

25.7 Producing PDF Output with the FOP Serializer

Using the XSQL pages framework support for custom serializers, the `oracle.xml.xsql.serializers.XSQLFOPSerializer` class provides integration with the Apache Formatting Objects Processor (FOP). The FOP processor renders a PDF document from an XML document containing XSL Formatting Objects.

As described in Table 24-1, the demo directory includes the `emptablefo.xsl` stylesheet and `emptable.xsql` page as illustrations. If you get an error trying to use the FOP serializer, then probably you do not have all of the required JAR files in the `CLASSPATH`. The `XSQLFOPSerializer` class resides in the separate `xml.jar` file, which must be included in the `CLASSPATH` to use the FOP integration. You must also add these additional Java archives to your `CLASSPATH`:

- `fop.jar`—from Apache, version 0.20.3 or later
- `batik.jar`—from the FOP distribution
- `avalon-framework-4.0.jar`—from FOP distribution
- `logkit-1.0.jar`—from FOP distribution

In case you want to customize the implementation, the source code for the FOP serializer provided in this release is shown in Example 25-18.

Copyright © 1999, 2025, Oracle and/or its affiliates.

> **See Also**
>
> *Apache FOP*

Example 25-18 Source Code for FOP Serializer

```
package oracle.xml.xsql.serializers;
import org.w3c.dom.Document;
import org.apache.log.Logger;
import org.apache.log.Hierarchy;
import org.apache.fop.messaging.MessageHandler;
import org.apache.log.LogTarget;
import oracle.xml.xsql.XSQLPageRequest;
import oracle.xml.xsql.XSQLDocumentSerializer;
import org.apache.fop.apps.Driver;
import org.apache.log.output.NullOutputLogTarget;
/**
 * Tested with the FOP 0.20.3RC release from 19-Jan-2002
 */
public class XSQLFOPSerializer implements XSQLDocumentSerializer {
  private static final String PDFMIME = "application/pdf";
  public void serialize(Document doc, XSQLPageRequest env) throws Throwable {
    try {
      // First make sure we can load the driver
      Driver FOPDriver = new Driver();
      // Tell FOP not to spit out any messages by default.
      // You can modify this code to create your own FOP Serializer
      // that logs the output to one of many different logger targets
      // using the Apache LogKit API
      Logger logger=Hierarchy.getDefaultHierarchy().getLoggerFor("XSQLServlet");
      logger.setLogTargets(new LogTarget[]{new NullOutputLogTarget()});
      FOPDriver.setLogger(logger);
      // Some of FOP's messages appear to still use MessageHandler.
      MessageHandler.setOutputMethod(MessageHandler.NONE);
      // Then set the content type before getting the reader
      env.setContentType(PDFMIME);
      FOPDriver.setOutputStream(env.getOutputStream());
      FOPDriver.setRenderer(FOPDriver.RENDER_PDF); FOPDriver.render(doc);
    }
    catch (Exception e) {
      // Cannot write PDF output for the error anyway.
      // So maybe this stack trace will be useful info
      e.printStackTrace(System.err);
    }
  }
}
```

25.8 Performing XSQL Customizations

XSQL customization topics are presented.

25.8.1 Writing Custom XSQL Action Handlers

The XSQL pages engine processes an XSQL page by looking for action elements from the `xsql` namespace and invoking an appropriate action element handler class to process each

Copyright © 1999, 2025, Oracle and/or its affiliates.

action. The processor supports any action that implements the `XSQLActionHandler` interface. All of the built-in actions implement this interface.

When a task requires custom processing, and none of the built-in actions listed in Table 33-2 does exactly what you need, you can write your own actions.

The XSQL engine processes the actions in a page in the following way. For each action in the page, the engine performs these steps:

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 can map the XSQL action element name to the Java class that implements the handler of the action. Table 33-2 lists the built-in actions and their corresponding classes.

For user-defined actions, use this built-in action, replacing `fully.qualified.Classname` with the name of your class:

```
<xsql:action handler="fully.qualified.Classname" ... />
```

The `handler` attribute provides the fully qualified name of the Java class that implements the custom action handler.

25.8.1.1 Implementing the XSQLActionHandler Interface

To create a custom action handler, provide a class that implements To create a custom action handler, provide a class that implements `oracle.xml.xsql.XSQLActionHandler` interface `oracle.xml.xsql.XSQLActionHandler`. Most custom action handlers extend `oracle.xml.xsql.XSQLActionHandlerImpl`, which provides a default implementation of the `init()` method and offers useful helper methods.

When an action handler's `handleAction()` method is invoked by the XSQL pages processor, a DOM fragment is passed to the action implementation. The action handler appends any dynamically created XML content returned to the page to the root node.

The XSQL processor conceptually replaces the action element in the XSQL page with the content of this document fragment. It is legal for an action handler to append nothing to this fragment if it has no XML content to add to the page.

While writing your custom action handlers, some methods on the `XSQLActionHandlerImpl` class are helpful. Table 25-2 lists these methods.

Table 25-2 Helpful Methods in the XSQLActionHandlerImpl Class

Method Name	Description
`getActionElement`	Returns the current action element being handled.
`getActionElementContent`	Returns the text content of the current action element, with all lexical parameters substituted appropriately.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 25-2 (Cont.) Helpful Methods in the XSQLActionHandlerImpl Class

Method Name	Description
`getPageRequest`	Returns the current XSQL pages processor context. Using this object you do this: • `setPageParam()` Set a page parameter value. • `getPostedDocument()/setPostedDocument()` Get or set the posted XML document. • `translateURL()` Translate a relative URL to an absolute URL. • `getRequestObject()/setRequestObject()` Get or set objects in the page request context that can be shared across actions in a single page. • `getJDBCConnection()` Gets the JDBC connection in use by this page (possible null if no connection in use). • `getRequestType()` 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 `XSQLPageRequest` object to the more specific `XSQLServletPageRequest` to access servlet-specific methods such as `getHttpServletRequest`, `getHttpServletResponse`, and `getServletContext`.
`getAttributeAllowingParam`	Retrieves the attribute value from an element, resolving any XSQL lexical parameter references that might appear in value of the attribute. Typically this method is applied to the action element itself, but it is also useful for accessing attributes of subelements. To access an attribute value without allowing lexical parameters, use the standard `getAttribute()` method on the DOM `Element` interface.
`appendSecondaryDocument`	Appends the contents of an external XML document to the root of the action handler result content.
`addResultElement`	Simplifies appending a single element with text content to the root of the action handler result content.
`firstColumnOfFirstRow`	Returns the first column value of the first row of a SQL statement. Requires the current page to have a connection attribute on its document element, or an error is returned.
`getBindVariableCount`	Returns the number of tokens in the space-delimited list of `bind-params`. This number indicates how many bind variables are expected to be bound to parameters.
`handleBindVariables`	Manages the binding of JDBC bind variables that appear in a prepared statement with the parameter values specified in the `bind-params` attribute on the current action element. If the statement is already using several bind variables before invoking this method, you can pass the number of existing bind variable slots in use as well.
`reportErrorIncludingStatement`	Reports an error. The error includes the offending (SQL) statement that caused the problem and optionally includes a numeric error code.
`reportFatalError`	Reports a fatal error.
`reportMissingAttribute`	Reports an error that a required action handler attribute is missing by using the `<xsql-error>` element.
`reportStatus`	Reports action handler status by using the `<xsql-status>` element.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 25-2 (Cont.) Helpful Methods in the XSQLActionHandlerImpl Class

Method Name	Description
`requiredConnectionProvided`	Checks whether a connection is available for this request and outputs an errorgram into the page if no connection is available.
`variableValue`	Returns the value of a lexical parameter, taking into account all scoping rules that might determine its default value.

Example 25-19 shows a custom action handler named `MyIncludeXSQLHandler` that leverages a built-in action handler. It uses arbitrary Java code to modify the XML fragment returned by this handler before appending its result to the XSQL page.

You might have to write custom action handlers that work differently based on whether the page is requested through the XSQL servlet, the XSQL command-line utility, or programmatically through the `XSQLRequest` class.You can invoke `getPageRequest()` in your action handler implementation to get a reference to the `XSQLPageRequest` interface for the current page request. By invoking `getRequestType()` on the `XSQLPageRequest` object, you can determine whether the request is coming from the Servlet, Command Line, or Programmatic routes. If the return value is `Servlet`, then you can access the HTTP servlet request, response, and servlet context objects as shown in Example 25-20.

Example 25-19 MyIncludeXSQLHandler.java

```
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);
  }
}
```

Example 25-20 Testing for the Servlet Request

```
XSQLServletPageRequest xspr = (XSQLServletPageRequest)getPageRequest();
if (xspr.getRequestType().equals("Servlet")) {
  HttpServletRequest     req  = xspr.getHttpServletRequest();
  HttpServletResponse   resp  = xspr.getHttpServletResponse();
  ServletContext        cont  = xspr.getServletContext();
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
  // Do something here with req, resp, or cont. Note that writing to the response
  // directly from a handler produces unexpected results. All the servlet or your
  // custom Serializer to write to the servlet response output stream at the right
  // moment later when all action elements have been processed.
}
```

25.8.1.2 Using Multivalued Parameters in Custom XSQL Actions

`XSQLActionHandlerImpl` is the base class for custom XSQL actions.

It supports:

- Array-named lexical parameter substitution
- Array-named bind variables
- Simple-valued parameters

If your custom actions use methods such as `getAttributeAllowingParam()`, `getActionElementContent()`, or `handleBindVariables()` from this base class, you pick up multivalued parameter functionality for free in your custom actions.

Use the `getParameterValues()` method on the `XSQLPageRequest` interface to explicitly get a parameter value as a `String[]`. The helper method `variableValues()` in `XSQLActionHandlerImpl` enables you to use this functionality from within a custom action handler if you must do so programmatically.

25.8.2 Implementing Custom XSQL Serializers

You can implement a user-defined serializer class to control how the final XSQL datapage is serialized to a text or binary stream. A user-defined serializer must implement interface `oracle.xml.xsql.XSQLDocumentSerializer`.

Interface `oracle.xml.xsql.XSQLDocumentSerializer` contains this single method:

```
void serialize(org.w3c.dom.Document doc, XSQLPageRequest env) throws
Throwable;
```

Only DOM-based serializers are supported. A custom serializer class is expected to perform these steps:

1. Set the content type of the serialized stream before writing any content to the output `PrintWriter` (or `OutputStream`).

   Set the type by invoking `setContentType()` on the `XSQLPageRequest` passed to your serializer. When setting the content type, you can set a MIME type:

   ```
   env.setContentType("text/html");
   ```

   Alternatively, you can set a MIME type with an explicit output encoding character set:

   ```
   env.setContentType("text/html;charset=Shift_JIS");
   ```

2. Invoke either `getWriter()` or `getOutputStream()` (but not both) on the `XSQLPageRequest` to get the appropriate `PrintWriter` or `OutputStream` for serializing the content.

The custom serializer in Example 25-21 shows a simple implementation that serializes an HTML document containing the name of the document element of the current XSQL data page.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Example 25-21 Custom Serializer

```
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>");
  }
}
```

25.8.2.1 Techniques for Using a Custom Serializer

There are two ways to use a custom serializer, depending on whether you must first perform an XSLT transformation before serializing.

To perform an XSLT transformation before using a custom serializer, add the `serializer="java:fully.qualified.ClassName"` in the `<?xml-stylesheet?>` processing instruction at the top of your page. The following examples shows this technique:

```
<?xml version="1.0?>
<?xml-stylesheet type="text/xsl" href="mystyle.xsl"
                 serializer="java:my.pkg.MySerializer"?>
```

If you need only the custom serializer, omit the `type` and `href` attributes. The following example shows this technique:

```
<?xml version="1.0?>
<?xml-stylesheet serializer="java:my.pkg.MySerializer"?>
```

25.8.2.2 Assigning a Short Name to a Custom Serializer

You can assign a short name to your custom serializers in the `<serializerdefs>` section of the XSQL configuration file. You can then use the short name in the serializer attribute, to save typing. The short name is case-sensitive.

Assume that you have the information shown in Example 25-22 in your XSQL configuration file. You can use the short names "Sample" or "FOP" in a stylesheet instruction:

```
<?xml-stylesheet type="text/xsl" href="emp-to-xslfo.xsl" serializer="FOP"?>
<?xml-stylesheet serializer="Sample"?>
```

The `XSQLPageRequest` interface supports both a `getWriter()` and a `getOutputStream()` method. Custom serializers can invoke `getOutputStream()` to return an `OutputStream`

Copyright © 1999, 2025, Oracle and/or its affiliates.

instance into which binary data can be serialized. When you use the XSQL servlet, writing to this output stream writes binary information to the servlet output stream.

The serializer shown in Example 25-23 shows an example of writing a dynamic GIF image. In this example the GIF image is a static "ok" icon, but it shows the basic technique that a more sophisticated image serializer must use.

Using the XSQL command-line utility, the binary information is written to the target output file. Using the `XSQLRequest` API, two constructors exist that allow the caller to supply the target `OutputStream` to use for the results of page processing.

Your serializer must either invoke `getWriter()` for textual output or `getOutputStream()` for binary output but not both. Invoking both in the same request raises an error.

Example 25-22 Assigning Short Names to Custom Serializers

```
<XSQLConfig>
  <!--and so on. -->
  <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>
```

Example 25-23 Writing a Dynamic GIF Image

```
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)0xFFFFFF80,(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)0xFFFFFF8C,
      (byte)0xF,(byte)0xFFFFFFA7,(byte)0xFFFFFFB8,(byte)0xFFFFFF9B,
      (byte)0xA,(byte)0xFFFFFFA2,(byte)0x79,(byte)0xFFFFFFE9,
      (byte)0xFFFFFF85,(byte)0x7A,(byte)0x27,(byte)0xFFFFFF93,
      (byte)0x5A,(byte)0xFFFFFFE3,(byte)0xFFFFFFEC,(byte)0x75,
      (byte)0x11,(byte)0xFFFFFF85,(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();
  }
}
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

25.8.3 Using a Custom XSQL Connection Manager for JDBC Data Sources

As an alternative to defining your named connections in the XSQL configuration file, you can use one of two `XSQLConnectionManager` implementations provided. These let you use your servlet container's JDBC data source implementation and related connection pooling features.

This XSQL pages framework provides this alternative connection manager implementations:

- `oracle.xml.xsql.XSQLDatasourceConnectionManager`

  Consider using this connection manager if your servlet container's data source implementation does *not* use the Oracle JDBC driver. Features of the XSQL pages system such as `<xsql:ref-cursor-function>` and `<xsql:include-owa>` are not available when you do not use an Oracle JDBC driver.

- `oracle.xml.xsql.XSQLOracleDatasourceConnectionManager`

  Consider using this connection manager when your data source implementation returns JDBC `PreparedStatement` and `CallableStatement` objects that implement the `oracle.jdbc.PreparedStatement` and `oracle.jdbc.CallableStatement` interfaces. The Oracle WebLogic Server has a data source implementation that performs this task.

When using either of the preceding alternative connection manager implementations, the value of the connection attribute in your XSQL page template is the Java Naming and Directory Interface (JNDI) name used to look up your desired data source. For example, the value of the connection attribute might look like:

- `jdbc/scottDS`
- `java:comp/env/jdbc/MyDatasource`

If you are not using the default XSQL pages connection manager, then needed connection pooling functionality must be provided by the alternative connection manager implementation. In the case of the preceding two options based on JDBC data sources, you must properly configure your servlet container to supply the connection pooling. See your servlet container documentation for instructions on how to properly configure the data sources to offer pooled connections.

25.8.4 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 perform these steps:

1. Write a connection manager factory class that implements the `oracle.xml.xsql.XSQLConnectionManagerFactory` interface.
2. Write a connection manager class that implements the `oracle.xml.xsql.XSQLConnectionManager` interface.
3. Change the name of the `XSQLConnectionManagerFactory` class in your XSQL configuration file.

The XSQL servlet uses your connection management scheme instead of the XSQL pages default scheme.

Copyright © 1999, 2025, Oracle and/or its affiliates.

You can set your custom connection manager factory as the default connection manager factory by providing the class name in the XSQL configuration file. Set the factory in this section:

```
<!--
 | Set the name of the XSQL Connection Manager Factory
 | implementation. The class must implement the
 | oracle.xml.xsql.XSQLConnectionManagerFactory interface.
 | If unset, the default is to use the built-in connection
 | manager implementation in
 | oracle.xml.xsql.XSQLConnectionManagerFactoryImpl
+-->
  <connection-manager>
      <factory>oracle.xml.xsql.XSQLConnectionManagerFactoryImpl</factory>
  </connection-manager>
```

In addition to specifying the default connection manager factory, you can associate a custom connection factory with a `XSQLRequest` object by using APIs provided.

The responsibility of the `XSQLConnectionManagerFactory` is to return an instance of an `XSQLConnectionManager` for use by the current request. In a multithreaded environment such as a servlet engine, the `XSQLConnectionManager` object must ensure that a single `XSQLConnection` instance is not used by two different threads. This aim is realized by marking the connection as in use for the time between the `getConnection()` and `releaseConnection()` method invocations. The default XSQL connection manager implementation automatically pools named connections and adheres to this thread-safe policy.

If your custom implementation of `XSQLConnectionManager` implements the optional `oracle.xml.xsql.XSQLConnectionManagerCleanup` interface, then your connection manager can clean up any resources it has allocated. For example, if your servlet container invokes the `destroy()` method on the `XSQLServlet` servlet, which can occur during online administration of the servlet for example, the connection manager has a chance to clean up resources as part of the servlet destruction process.

25.8.4.1 Accessing Authentication Information in a Custom Connection Manager

To use the HTTP authentication mechanism to get the user name and password to connect to the database, write a customized connection manager. You can then invoke a `getConnection()` method to get the needed information.

You can write a Java program that follows these steps:

1. Pass an instance of the `oracle.xml.xsql.XSQLPageRequest` interface to the `getConnection()` method.
2. Invoke `getRequestType()` to ensure that the request type is `Servlet`.
3. Cast the `XSQLPageRequest` object to an `XSQLServletPageRequest`.
4. Invoke `getHttpServletRequest()` on the result of the preceding step.
5. Get the authentication information from the `javax.servlet.http.HttpServletResponse` object returned by the previous invocation.

Copyright © 1999, 2025, Oracle and/or its affiliates.

25.8.5 Implementing a Custom XSQLErrorHandler

You can control how serious page processor errors such as an unavailable connection are reported to users by implementing interface `oracle.xml.xsql.XSQLErrorHandler`.

The interface contains this single method signature:

```
public interface XSQLErrorHandler {
  public void handleError( XSQLError err, XSQLPageRequest env);
}
```

You can provide a class that implements the `XSQLErrorHandler` interface to customize how the XSQL pages processor writes error messages. The new `XSQLError` object encapsulates the error information and provides access to the error code, formatted error message, and so on.

Example 25-24 shows a sample implementation of `XSQLErrorHandler`.

You can control which custom `XSQLErrorHandler` implementation is used in these distinct ways:

- Define the name of a custom `XSQLErrorHandler` implementation class in the XSQL configuration file. You must provide the fully qualified class name of your error handler class as the value of the `/XSQLConfig/processor/error-handler/class` entry.

  If the XSQL processor can load this class, and if it correctly implements the `XSQLErrorHandler` interface, then it uses this class as a singleton and replaces the default implementation globally wherever page processor errors are reported.
- Override the error writer on a per page basis by using the `errorHandler` (or `xsql:errorHandler`) attribute on the document element of the page. The attribute value is the fully qualified class name of a class that implements the `XSQLErrorHandler` interface. This class reports the errors only for this page. The class is instantiated on each page request by the page engine.

You can use a combination of the preceding approaches if needed.

Example 25-24 myErrorHandler class

```
package example;
import oracle.xml.xsql.*;
import java.io.*;
public class myErrorHandler implements XSQLErrorHandler {
  public void logError( XSQLError err, XSQLPageRequest env) {
    // Must set the content type before writing anything out
    env.setContentType("text/html");
    PrintWriter pw = env.getErrorWriter();
    pw.println("<H1>ERROR</H1><hr>"+err.getMessage());
  }
}
```

25.8.6 Providing a Custom XSQL Logger Implementation

You can optionally register custom code to handle the logging of the start and end of each XSQL page request. Your custom logger code must provide an implementation of interfaces `oracle.xml.xsql.XSQLLoggerFactory` and `oracle.xml.xsql.XSQLLogger`.

The `XSQLLoggerFactory` interface contains this single method:

Copyright © 1999, 2025, Oracle and/or its affiliates.

```
public interface XSQLLoggerFactory {
  public XSQLLogger create( XSQLPageRequest env);
}
```

You can provide a class that implements the `XSQLLoggerFactory` interface to decide how `XSQLLogger` objects are created (or reused) for logging. The XSQL processor holds a reference to the `XSQLLogger` object returned by the factory for the duration of a page request. The processor uses it to log the start and end of each page request by invoking the `logRequestStart()` and `logRequestEnd()` methods.

The `XSQLLogger` interface is:

```
public interface XSQLLogger {
   public void logRequestStart(XSQLPageRequest env) ;
   public void logRequestEnd(XSQLPageRequest env);
}
```

The classes in Example 25-25 and Example 25-26 show a trivial implementation of a custom logger. The `XSQLLogger` implementation in Example 25-25 notes the time the page request started. It then logs the page request end by printing the name of the page request and the elapsed time to `System.out`.

The factory implementation is shown in Example 25-26.

To register a custom logger factory, edit the `XSQLConfig.xml` file and provide the name of your custom logger factory class as the content to the `/XSQLConfig/processor/logger/factory` element. Example 25-27 shows this technique.

By default, `<logger>` section is commented out. There is no default logger.

Example 25-25 SampleCustomLogger Class

```
package example;
import oracle.xml.xsql.*;
public class SampleCustomLogger implements XSQLLogger  {
  long start = 0;
  public void logRequestStart(XSQLPageRequest env) {
    start = System.currentTimeMillis();
  }
  public void logRequestEnd(XSQLPageRequest env) {
    long secs = System.currentTimeMillis() - start;
    System.out.println("Request for " + env.getSourceDocumentURI()
                       + " took "+ secs + "ms");
  }
}
```

Example 25-26 SampleCustomLoggerFactory Class

```
package example;
import oracle.xml.xsql.*;
public class SampleCustomLoggerFactory implements XSQLLoggerFactory {
  public XSQLLogger create(XSQLPageRequest env) {
    return new SampleCustomLogger();
  }
}
```

Example 25-27 Registering a Custom Logger Factory

```
<XSQLConfig>
    :
  <processor>
        :
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
      <logger>
         <factory>example.SampleCustomLoggerFactory</factory>
      </logger>
         :
   </processor>
</XSQLConfig>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

Part III

Oracle XML Developer's Kit for C++

An explanation is given of how to use Oracle XML Developer's Kit (XDK) to develop C++ applications.

Copyright © 1999, 2025, Oracle and/or its affiliates.

26

Getting Started with Oracle XML Developer's Kit for C++

An explanation is given of how to get started with Oracle XML Developer's Kit (XDK) for C++. The C++ demo programs are on the Oracle Database Examples media.

26.1 Installing XDK for C++ Components

The XDK for C++ components are included with Oracle Database.

Related Topics

- About Installing XDK
  The standard installation of Oracle Database includes XDK (all of its components).

> **See Also**
>
> Overview of XDK for a list of the XDK for C++ components

26.2 Configuring the UNIX Environment for XDK for C++ Components

Topics here include component dependencies, environment variables, the runtime and compile-time environments, and the component version.

26.2.1 XDK for C++ Component Dependencies on UNIX

The C++ libraries described in this section are located in `$ORACLE_HOME/lib`.

The XDK for C and C++ components are contained in the library:

```
libxml21.a
```

In addition to the XDK for C components described in XDK for C Component Dependencies on UNIX, the library includes the XML class generator, which creates C++ source files based on an input document type definition (DTD) or XML Schema.

Table 3-1 in XDK for C Component Dependencies on UNIX describes the Oracle CORE and Globalization Support libraries on which the XDK for C components (UNIX) depend. The library dependencies are the same for C and C++.

Copyright © 1999, 2025, Oracle and/or its affiliates.

26.2.2 Setting Up XDK for C++ Environment Variables on UNIX

The UNIX environment variables required for use with the XDK components are the same for C and C++.

See Table 3-2 in Setting Up XDK for C Environment Variables on UNIX.

26.2.3 Testing the XDK for C++ Runtime Environment on UNIX

You can test your environment by running any of the XDK C utilities for UNIX. These utilities do not have C++ versions.

The C utilities described in Table 3-3 in Testing the XDK for C Runtime Environment on UNIX.

26.2.4 Setting Up and Testing the XDK for C++ Compile-Time Environment on UNIX

How to set up and test the XDK C++ compile-time UNIX environment is described.

Both the C and C++ header files are located in `$ORACLE_HOME/xdk/include`. Table 26-1 describes the C++ header files. Table 3-4 in Setting Up and Testing the XDK C Compile-Time Environment on UNIX describes the C header files. Your runtime environment must be set up before you can compile your C++ code.

Table 26-1 Header Files in the XDK for C++ Compile-Time Environment

Header File	Description
`oraxml.hpp`	Includes the Oracle9i XML ORA data types and the public ORA application programming interfaces (APIs) included in `libxml.a` (only for backward compatibility).
`oraxmlcg.h`	Includes the C APIs for the C++ class generator (only for backward compatibility).
`oraxsd.hpp`	Includes the Oracle9i XML schema definition (XSD) validator data types and APIs (only for backward compatibility)
`xml.hpp`	Handles the Unified Document Object Model (DOM) APIs transparently, whether you use them through Oracle Call Interface (OCI) or standalone
`xmlotn.hpp`	Includes the common APIs, whether you compile standalone or use OCI and the Unified DOM
`xmlctx.hpp`	Includes the initialization and exception-handling public APIs

26.2.4.1 Testing the XDK for C++ Compile-Time Environment on UNIX

The simplest way to test your compile-time environment is to run the `make` utility on the sample programs.

The demo programs are located on the Examples media rather than the Oracle Database CD. After installing these programs, they are located in `$ORACLE_HOME/xdk/demo/cpp`.

Build and run the sample programs by executing these commands at the system prompt:

```
cd $ORACLE_HOME/xdk/demo/cpp
make
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

26.2.5 Verifying the XDK for C++ Component Version on UNIX

How to determine which version of XDK you have is explained.

To get the version of XDK that you are using, change into `$ORACLE_HOME/lib` and run this command as the system prompt:

```
strings libxml21.a | grep -i developers
```

26.3 Configuring the Windows Environment for XDK for C++ Components

Topics here include component dependencies, environment variables, testing the runtime environment, setting up and testing the compile-time environment, and Visual C/C++.

26.3.1 XDK for C++ Component Dependencies on Windows

The C++ libraries described in this section are located in `%ORACLE_HOME%\bin`.

The XDK for C and C++ components are contained in this Windows library:

```
libxml21.dll
```

Table 3-5 in XDK for C Component Dependencies on Windows describes the Oracle Common Oracle Runtime Environment (CORE) and Globalization Support libraries on which the C components for Windows depend. The library dependencies are the same for C and C++.

26.3.2 Setting Up XDK for C++ Environment Variables on Windows

The Windows environment variables required for use with the XDK are the same for C and C++.

See Table 3-6 in Setting Up XDK for C Environment Variables on Windows.

26.3.3 Testing the XDK for C++ Runtime Environment on Windows

You can test your environment by running any of the XDK C utilities for UNIX. These utilities do not have C++ versions.

These utilities are described in Table 3-7 in Testing the XDK for C Runtime Environment on Windows.

26.3.4 Setting Up and Testing the XDK for C++ Compile-Time Environment on Windows

How to set up and test the XDK C++ compile-time Microsoft Windows environment is described.

Table 26-1 in the section Setting Up and Testing the XDK for C++ Compile-Time Environment on UNIX describes the header files required for compilation of the C components on Windows. The relative file names are the same on both UNIX and Windows installations.

Copyright © 1999, 2025, Oracle and/or its affiliates.

On Windows the header files are located in `%ORACLE_HOME%\xdk\include`. Your runtime environment must be set up before you can compile your code.

26.3.4.1 Testing the XDK for C++ Compile-Time Environment on Windows

You can test your compile-time environment by compiling the demo programs, which are located in `%ORACLE_HOME%\xdk\demo\cpp` if you have installed the Oracle Database Examples media.

The procedure for setting the C++ compiler path is identical to the procedure described in Setting the XDK for C Compiler Path on Windows. The procedure for editing the `Make.bat` files is identical to the procedure described in Editing the Make.bat Files on Windows.

26.3.5 Using the XDK for C++ Components with Visual C/C++

You can set up a project in Microsoft Visual C/C++ and use it for the demos included in XDK.

See Using the XDK for C Components and Visual C++ in Microsoft Visual Studio for instructions.

Copyright © 1999, 2025, Oracle and/or its affiliates.

27

Overview of the Unified C++ Interfaces

The unified C++ interfaces are described.

27.1 What Is the Unified C++ API?

Unified C++ application programming interfaces (APIs) for Extensible Markup Language (XML) tools represent a set of C++ interfaces for Oracle XML tools. All three kinds of C++ interfaces: abstract classes, templates, and implicit interfaces represented by generic template parameters, are used by the unified framework.

This unified approach provides a generic, interface-based framework that allows XML tools to be improved, updated, replaced, or added without affecting any interface-based user code, and minimally affecting application drivers and, possibly, application configuration.

> **Note**
>
> Use the unified C++ API in `xml.hpp` for Oracle XML Developer's Kit (XDK) applications. The older, nonunified C++ API in `oraxml.hpp` is deprecated and supported only for backward compatibility. It will be removed in a future release.
>
> The unified C++ API supports the World Wide Web Consortium (W3C) specification as closely as possible. However, Oracle cannot guarantee that the specification is fully supported by our implementation because the W3C specification does not cover C++ implementations.

27.2 Accessing the C++ Interface

The C++ interface is provided with Oracle Database. Sample files are located in `$ORACLE_HOME/xdk/demo/cpp`. `readme.html` in the root directory of the software archive contains release specific information including bug fixes and API additions.

27.3 OracleXML Namespace

`OracleXml` is the C++ namespace for all XML C++ interfaces. It contains common interfaces and namespaces for different XDK packages.

The following namespaces are included in namespace `OracleXML`:

- `Ctx`—namespace for TCtx related declarations
- `Dom`—namespace for Document Object Model (DOM) related declarations
- `Parser`—namespace for parser and schema validator declarations
- `IO`—namespace for input and output source declarations

Copyright © 1999, 2025, Oracle and/or its affiliates.

- `Xsl`—namespace for Extensible Stylesheet Language Transformation (XSLT) related declarations
- `XPath` - namespace for XPath related declarations
- `Tools`—namespace for Tools::Factory related declarations

`OracleXml` is fully defined in the file `xml.hpp`. Another namespace, `XmlCtxNS`, visible to users, is defined in `xmlctx.hpp`. That namespace contains C++ definitions of data structures corresponding to C level definitions of the `xmlctx` context and related data structures. While there is no need for users to know details of that namespace, `xmlctx.hpp` must be included in most application main modules.

Multiple encodings are currently supported on the base of the `oratext` type that is currently supposed to be used by all implementations. All strings are represented as `oratext*`.

27.3.1 OracleXML Interfaces

`XMLException` interface—This is the root interface for all XML exceptions.

27.4 Ctx Namespace

The Ctx namespace contains data types and interfaces related to the TCtx interface.

27.4.1 OracleXML Data Types

OracleXML data types are described.

`encoding`—a particular supported encoding. The following kinds of encodings (or encoding names) are supported:

- data_encoding
- default_input_encoding
- input_encoding—overwrites the previous one
- error_language—gets overwritten by the language of the error handler, if specified

`encodings`—array of encodings.

27.4.2 Ctx Interfaces

Ctx interfaces `ErrorHandler`, `MemAllocator`, and `Tctx` are described.

`ErrorHandler` Interface — This is the root error handler class. It deals with local processing of errors, mainly from the underlying C implementation. In some implementations, it might throw `XmlException`. To accommodate the needs of all implementations, this behavior is not specified in its signature. However, it can create exception objects. The error handler is passed to the `TCtx` constructor when `TCtx` is initialized. Implementations of this interface are provided by the user.

`MemAllocator` Interface—This is a simple root interface to make the `TCtx` interface reasonably generic so that different allocator approaches can be used in the future. It is passed to the `TCtx` constructor when `TCtx` is initialized. It is a low level allocator that does not know the type of an object being allocated. The allocators with this interface can also be used directly. In this case the user is responsible for the explicit deallocation of objects (with `dealloc`).

Copyright © 1999, 2025, Oracle and/or its affiliates.

If the `MemAllocator` interface is passed as a parameter to the `TCtx` constructor, it often makes sense to overwrite the operator new. In this case, all memory allocations in both C and C++ can be done by the same allocator.

`Tctx` Interface—This is an implicit interface to XML context implementations. It is primarily used for memory allocation, error (not exception) handling, and different encodings handling. The context interface is an implicit interface that is supposed to be used as type parameter. The name `TCtx` is used as a corresponding type parameter name. Its actual substitutions are instantiations of implementations parameterized (templatized) by real context implementations. In the case of errors `XmlException` might be thrown.All constructors create and initialize context implementations. In a multithreaded environment a separate context implementation must be initialized for each thread.

27.5 IO Namespace

The `IO` namespace specifies interfaces for the different input and output options for all XML tools.

27.5.1 IO Data Types

Data type `InputSourceType` specifies the kinds of input sources that are supported.

- `ISRC_URI`—Input is to be read from the specified Universal Resource Identifier (URI).
- `ISRC_FILE`—Input is to be read from a file.
- `ISRC_BUFFER`—Input is to be read from a buffer.
- `ISRC_DOM`—Input is a DOM tree.
- `ISRC_CSTREAM`—Input is a C level stream.

27.5.2 IO Interfaces

The interfaces to inputs are described.

`URISource`—This is an interface to inputs from specified URIs.

`FileSource`—This is an interface to inputs from a file.

`BufferSource`—This is an interface to inputs from a buffer.

`DOMSource`—This is an interface to inputs from a DOM tree.

`CStreamSource`—This is an interface to inputs from a C level stream.

27.6 Tools Package

`Tools` is the package (subspace of `OracleXml`) for types and interfaces that are related to the creation and instantiation of Oracle XML tools.

27.6.1 Tools Interfaces

Interfaces for XML tools are described.

`FactoryException`—Specifies tools factory exceptions. It is derived from `XMLExceptions`.

Copyright © 1999, 2025, Oracle and/or its affiliates.

`Factory`—XML tools factory. Hides implementations of all XML tools and provides methods to create objects representing these tools based on their identifier (ID) values.

27.7 Error Message Files

Error message files are provided in directory `$ORACLE_HOME/xdk/mesg`. You can set environment variable `ORA_XML_MESG` to point to this directory, but this not required.

> **See Also**
>
> *Oracle AI Database XML C++ API Reference* package Ctx APIs for C++

Copyright © 1999, 2025, Oracle and/or its affiliates.

28

Using the XML Parser for C++

An explanation is given of how to use the Extensible Markup Language (XML) parser for C++.

> **Note**
>
> Use the unified C++ application programming interface (API) in `xml.hpp` for Oracle XML Developer's Kit (XDK) applications. The older, nonunified C++ API in `oraxml.hpp` is deprecated and supported only for backward compatibility. It will be removed in a future release.

28.1 Introduction to Oracle XML Parser for C++

Oracle XML parser for C++ determines whether an XML document is well-formed and optionally validates it against a document type definition (DTD) or Extensible Markup Language (XML) schema. The parser constructs an object tree that can be accessed through one of these two XML APIs:

- Document Object Model (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 DOM, a standard tree-based API for XML and HTML documents.
- Simple API for XML (SAX): Event-based APIs. An event-based API reports parsing events (such as the start and end of elements) directly to the application through a user defined SAX even handler, 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 must 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.

28.2 DOM Namespace

The DOM namespace is the namespace for DOM-related types and interfaces.

DOM interfaces are represented as generic references to different implementations of the DOM specification. They are parameterized by `Node` that supports various specializations and instantiations. Of them, the most important is `xmlnode` which corresponds to the current C implementation

These generic references do not have a `NULL`-like value. Any implementation must never create a reference with no state (like `NULL`). If it is necessary to signal that something has no state, the implementation must throw an exception.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Many methods might throw the SYNTAX_ERR exception, if the DOM tree is incorrectly formed, or they might throw UNDEFINED_ERR, when encountering incorrect parameters or unexpected NULL pointers. If these are the only errors that a particular method might throw, it is not reflected in the method signature.

Actual DOM trees do *not* depend on the context, TCtx. However, manipulations on DOM trees in the current, xmlctx-based implementation require access to the current context, TCtx. This is accomplished by passing the context pointer to the constructor of DOMImplRef. In multithreaded environment DOMImplRef is always created in the thread context and, so, has the pointer to the right context.

DOMImplRef provides a way to create DOM trees. DomImplRef is a reference to the actual DOMImplementation object that is created when a regular, noncopy constructor of DomImplRef is invoked. This works well in a multithreaded environment where DOM trees must be shared, and each thread has a separate TCtx associated with it. This works equally well in a single threaded environment.

DOMString is one encoding supported by Oracle implementations. The support of other encodings is an Oracle extension. The oratext* data type is used for all encodings. Interfaces represent DOM level 2 Core interfaces according to *Document Object Model Core*. These C++ interfaces support the DOM specification as closely as possible. However, Oracle cannot guarantee that the specification is fully supported by our implementation because the World Wide Web Consortium (W3C) specification does not cover C++ binding.

28.2.1 DOM Data Types

DomNodeType defines types of DOM nodes. DomExceptionCode defines exception codes returned by the DOM API.

28.2.2 DOM Interfaces

The DOM interfaces are described.

DOMException Interface—See exception DOMException in the W3C DOM documentation. DOM operations raise exceptions only in "exceptional" circumstances: when an operation is impossible to perform (either for logical reasons, because data is lost, or because the implementation has become unstable). The functionality of XMLException can be used for a wider range of exceptions.

NodeRef Interface—See interface Node in the W3C documentation.

DocumentRef Interface—See interface Document in the W3C documentation.

DocumentFragmentRef Interface—See interface DocumentFragment in the W3C documentation.

ElementRef Interface—See interface Element in the W3C documentation.

AttrRef Interface—See interface Attr in the W3C documentation.

CharacterDataRef Interface—See interface CharacterData in the W3C documentation.

TextRef Interface—See Text nodes in the W3C documentation.

CDATASectionRef Interface—See CDATASection nodes in the W3C documentation.

CommentRef Interface—See Comment nodes in the W3C documentation.

ProcessingInstructionRef Interface—See PI nodes in the W3C documentation.

Copyright © 1999, 2025, Oracle and/or its affiliates.

`EntityRef` Interface—See `Entity` nodes in the W3C documentation.

`EntityReferenceRef` Interface—See `EntityReference` nodes in the W3C documentation.

`NotationRef` Interface—See `Notation` nodes in the W3C documentation.

`DocumentTypeRef` Interface—See `DTD` nodes in the W3C documentation.

`DOMImplRef` Interface—See interface `DOMImplementation` in the W3C DOM documentation. `DOMImplementation` is fundamental for manipulating DOM trees. Every DOM tree is attached to a particular DOM implementation object. Several DOM trees can be attached to the same DOM implementation object. Each DOM tree can be deleted and deallocated by deleting the document object. All DOM trees attached to a particular DOM implementation object are deleted when this object is deleted. The `DOMImplementation` object is not visible to the user directly. It is visible through the class `DOMImplRef`. This functionality is needed because of requirements for multithreaded environments.

`NodeListRef` Interface—Abstract implementation of node list. See interface NodeList in the W3C documentation.

`NamedNodeMapRef` Interface—Abstract implementation of a node map. See interface NamedNodeMap in the W3C documentation.

28.2.3 DOM Traversal and Range Data Types

`AcceptNodeCode` is the data type for values returned by node filters for iterators and tree walkers. `WhatToShowCode` is the data type for codes to filter nodes. `RangeExceptionCode` is the data type for exceptions that can be thrown by interface `Range`. `CompareHowCode` is the data type for range comparisons.

28.2.4 DOM Traversal and Range Interfaces

The DOM 2 traversal and range interfaces are `NodeFilter`, `NodeIterator`, `TreeWalker`, `DocumentTraversal`, `RangeException`, `Range`, and `DocumentRange`.

28.3 Parser Namespace

Interfaces associated with the parser namespace are described.

`DOMParser` Interface—DOM parser root class.

`GParser` Interface—Root class for XML parsers.

`ParserException` Interface—Exception class for parser and validator.

`SAXHandler` Interface—Root class for current SAX handler implementations.

`SAXHandlerRoot` Interface—Root class for all SAX handlers.

`SAXParser` Interface—Root class for all SAX parsers.

`SchemaValidator` Interface—XML schema-aware validator.

28.3.1 GParser Interface

Interface `GParser` is the root class for all XML parser interfaces and implementations. It is not an abstract class; that is, it is not an interface. It is a real class that you can use to set and check parser parameters.

Copyright © 1999, 2025, Oracle and/or its affiliates.

28.3.2 DOMParser Interface

Interface `DOMParser` is the DOM parser root abstract class or interface. In addition to parsing and checking that a document is well formed, it provides ways to validate a document against a document type definition (DTD) or an XML schema.

28.3.3 SAXParser Interface

Interface `SAXParser` is the root abstract class for all SAX parsers.

28.3.3.1 SAX Event Handlers

To use SAX, a SAX event handler class must be provided by the user and passed to the SAXParser in a `parse()` invocation or set before such invocation.

`SAXHandlerRoot` Interface—root class for all SAX handlers.

`SAXHandler` Interface—root class for current SAX handler implementations.

28.4 Thread Safety for the XML Parser for C++

If threads are forked in the midst of the init–parse–term sequence of invocations, unpredictable behavior or results can occur.

28.5 XML Parser for C++ Usage

Invoke `Tools::Factory` to create a parser and initialize the parsing process. The XML input can be kind of `InputSource` (see `IO` namespace). `DOMParser` invocation produces the DOM tree. `SAXParser` invocation produces SAX events. Invoking the `parser` destructor terminates the process.

28.6 XML Parser for C++ Default Behavior

The default behavior for the XML parser for C++ is described.

- Character set encoding is 8-bit encoding of Unicode (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 specified.
- XML parser for C++ determines whether an XML document is well-formed and optionally validates it against a DTD. The parser constructs an object tree that can be accessed through a DOM interface or operates serially through a SAX interface.
- A parse tree which can be accessed by DOM APIs is built unless `saxcb` is set to use the SAX callback APIs. You can set any of the SAX callback functions 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 white space processing is to be fully conformant with the XML 1.0 spec, that is, all white space is reported back to the application but it is indicated which white space is ignorable. However, some applications may prefer to set the `XML_FLAG_DISCARD_WHITESPACE` which discards all white space between an end-element tag and this start-element tag.

Copyright © 1999, 2025, Oracle and/or its affiliates.

> **Note**
>
> Oracle recommends 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.

- 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.

28.7 C++ Sample Files

Directory `xdk/demo/cpp/parser/` contains several XML applications that show how to use the XML parser for C++ with the DOM and SAX interfaces.

Change directories to the sample directory (`$ORACLE_HOME/xdk/demo/cpp` on Solaris, for example) and read the `README` file. This document explains how to build the sample programs.

Table 28-1 lists the sample files in the directory. Each file `*Main.cpp` has a corresponding `*Gen.cpp` and `*Gen.hpp`.

Table 28-1 XML Parser for C++ Sample Files

Sample File Name	Description
`DOMSampleMain.cpp`	Sample usage of C++ interfaces of XML parser and DOM.
`FullDOMSampleMain.cpp`	Manually build DOM and then exercise.
`SAXSampleMain.cpp`	Source for SAXSample program.

> **See Also**
>
> *Oracle AI Database XML C++ API Reference* for parser package APIs for C++

Copyright © 1999, 2025, Oracle and/or its affiliates.

29
Using the XSLT Processor for C++

An explanation is given of how to use the Extensible Stylesheet Language Transformation (XSLT) processor for C++.

> **Note**
>
> Use the unified C++ application programming interface (API) in `xml.hpp` for Oracle XML Developer's Kit (XDK) applications. The older, nonunified C++ API in `oraxml.hpp` is deprecated and supported only for backward compatibility. It will be removed in a future release.

29.1 Accessing XSLT for C++

Extensible Stylesheet Language Transformation (XSLT) for C++ is provided with Oracle Database.

Sample files are located at `xdk/demo/cpp/new`.

`readme.html` in the root directory of the software archive contains release specific information including bug fixes and API additions.

Related Topics

- XSLT XVM Processor
  The Oracle XVM package includes the XSLT compiler and the XVM. This package implements the XSLT language as specified in the World Wide Web Consortium (W3C) Recommendation of 16 November 1999.

29.2 XSL Namespace

This is the namespace for XSLT compilers and transformers.

29.2.1 XSL Interfaces

The XSL interfaces are described.

`XslException` Interface—Root interface for all XSLT-related exceptions.

`Transformer` Interface—Basic XSLT processor. You can use this interface to invoke all XSLT processors.

`CompTransformer` Interface—Extended XSLT processor. You can use this interface only with processors that create intermediate binary bytecode (currently only the XVM-based processor).

`Compiler` Interface—XSLT compiler. It is used for compilers that compile XSLT into binary bytecode.

Copyright © 1999, 2025, Oracle and/or its affiliates.

> **See Also**
>
> *Oracle AI Database XML C++ API Reference* package XSL APIs for C++

29.3 XSLT for C++ DOM Interface Usage

Basic usage of XSLT for C++ DOM is described.

There are two inputs to `XMLParser.xmlparse()`:

- The Extensible Markup Language (XML) document
- The XSLT stylesheet to be applied to the XML document

An XSLT processor is initiated by invoking the tools factory to create a particular XSLT transformer or compiler.

An XSLT stylesheet is supplied to a transformer by invoking `setXSL()` member functions.

An XML instance document is supplied as a parameter to the transform member functions.

The resultant document (XML, HTML, Vector Markup Language (VML), and so on) is typically sent to an application for further processing. It is sent as a Document Object Model (DOM) tree or as a sequence of Simple API for XML (SAX) events. SAX events are produced if a SAX event handler is provided by the user.

The application terminates the XSLT processors by invoking their destructors.

29.4 Invoking XSLT for C++

You can invoke XSLT for C++ by invoking the executable on the command line or by writing C++ code and using the supplied APIs.

29.4.1 Command-Line Usage

XSLT for C++ can be called as an executable by invoking `bin/xml`.

> **See Also**
>
> Table 5-4

29.4.2 Writing C++ Code to Use Supplied APIs

XSLT for C++ can be invoked by writing code to use the supplied APIs. The code must be compiled using the headers in directory `public` and linked against the libraries in directory `lib` . See `Makefile` or `make.bat` in `xdk/demo/cpp/new` for full details of how to build your program.

Copyright © 1999, 2025, Oracle and/or its affiliates.

29.5 Using the Sample Files Included with the Software

Directory `$ORACLE_HOME/xdk/demo/cpp/parser/` contains several XML applications that show how to use the XSLT for C++.

Table 29-1 XSLT for C++ Sample Files

Sample File Name	Description

`XSLSampleMain.cpp`
`XSLSampleGen.cpp`
`XSLSampleGen.hpp`
	Sources for sample XSLT usage program. `XSLSample` takes two arguments, the XSLT stylesheet and the XML file. If you redirect `stdout` of this program to a file, you may have some output missing, depending on your environment.

`XVMSampleMain.cpp`
`XVMSampleGen.cpp`
`XVMSampleGen.hpp`
	Sources for the sample XSLT Virtual Machine (XVM) usage program.

Copyright © 1999, 2025, Oracle and/or its affiliates.

30

Using the XML Schema Processor for C++

An explanation is given of how to use the Extensible Markup Language (XML) schema processor for C++.

> **Note**
>
> Use the unified C++ application programming interface (API) in `xml.hpp` for Oracle XML Developer's Kit (XDK) applications. The older, nonunified C++ API in `oraxml.hpp` is deprecated and supported only for backward compatibility. It will be removed in a future release.

30.1 Oracle XML Schema Processor for C++

The XML Schema processor for C++ is a companion component to the Extensible Markup Language (XML) parser for C++ that allows support to simple and complex data types into XML applications.

The XML Schema processor for C++ supports the World Wide Web Consortium (W3C) XML Schema Recommendation. This makes writing custom applications that process XML documents straightforward, and means that a standards-compliant XML Schema processor is part of XDK on each operating system where Oracle Database is ported.

30.1.1 Oracle XML Schema for C++ Features

The features of the Oracle XML Schema processor for C++ are described.

These are the features:

- Supports simple and complex types
- Built upon the XML parser for C++
- Supports the W3C XML Schema Recommendation

The XML Schema processor for C++ class is `OracleXml::Parser::SchemaValidator`.

> **See Also**
>
> *Oracle AI Database XML C++ API Reference* schema validator interface

30.1.1.1 Online Documentation

Documentation for Oracle XML Schema processor for C++ is located in `/xdk/doc/cpp/schema` directory in your install area.

Copyright © 1999, 2025, Oracle and/or its affiliates.

30.1.2 Standards Conformance for Oracle XML Schema Processor for C++

The standards to which the XML Schema Processor for C++ conforms are listed.

- W3C recommendation for Extensible Markup Language (XML) 1.0
- W3C recommendation for Document Object Model Level 1.0
- W3C recommendation for Namespaces in XML 1.0
- W3C recommendation for XML Schema 1.0

30.2 XML Schema Processor API

Interface `SchemaValidator` is an abstract template class to handle XML schema-based validation of XML documents.

30.2.1 Invoking XML Schema Processor for C++

The XML Schema processor for C++ can be called as an executable by invoking `bin/schema` in the install area. This takes the arguments:

- XML instance document
- Optionally, a default schema
- Optionally, the working directory

Table 30-1 lists the options (can be listed if the option is invalid or -h is the option):

Table 30-1 XML Schema Processor for C++ Command-Line Options

Option	Description
`-0`	Always exit with code 0 (success).
`-e` *`encoding`*	Specify default input file encoding.
`-E` *`encoding`*	Specify output/data/presentation encoding.
`-h`	Help. Prints these choices.
`-i`	Ignore provided schema.
`-o` *`num`*	Validation option.
`-p`	Print document instance to `stdout` on success.
`-u`	Force the Unicode path.
`-v`	Version—display version, then exit.

The XML Schema processor for C++ 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` or `Make.bat` in the `xdk/demo/cpp/schema` directory for details on how to build your program.

Error message files in different languages are provided in the `mesg` subdirectory.

Copyright © 1999, 2025, Oracle and/or its affiliates.

30.3 Running the Provided XML Schema for C++ Sample Programs

Directory `$ORACLE_HOME/xdk/demo/cpp/schema` contains a sample application that shows how to use Oracle XML Schema processor for C++ with its API.

Table 30-2 lists the sample files provided.

Table 30-2 XML Schema Processor for C++ Samples Provided

Sample File	Description
`Makefile`	Makefile to build the sample programs and run them, verifying correct output.
`xsdtest.cpp`	Trivial program which invokes the XML Schema for C++ API
`car.{xsd,xml,std}`	Sample schema, instance document, expected output respectively, after running `xsdtest` on them.
`aq.{xsd,xml,std}`	Second sample schema, instance document, expected output, respectively, after running `xsdtest` on them.
`pub.{xsd,xml,std}`	Third sample schema, instance document, expected output respectively, after running `xsdtest` on them.

To build the sample programs, run `make`.

To build the programs and run them, comparing the actual output to expected output:

```
make sure
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

31
Using the XPath Processor for C++

An explanation is given of how to use the XPath processor for C++.

> **Note**
>
> Use the unified C++ application programming interface (API) in `xml.hpp` for Oracle XML Developer's Kit (XDK) applications. The older, nonunified C++ API in `oraxml.hpp` is deprecated and supported only for backward compatibility. It will be removed in a future release.

31.1 XPath Interfaces

The XPath interfaces are described.

`Processor` Interface—basic XPath processor interface to which any XPath processor must conform.

`CompProcessor` Interface—extended XPath processor that adds an ability to use XPath expressions precompiled into an internal binary representation. In this release this interface represents Oracle virtual machine interface.

`Compiler` Interface—XPath compiler into binary representation.

`NodeSetRef` Interface—defines references to node sets when they are returned by the XPath expression evaluation.

`XPathException` Interface—exceptions for XPath compilers and processors.

`XPathObject` Interface—interface for XPath 1.0 objects.

31.2 Sample Programs

Sample programs are located in `xdk/demo/cpp/new`.

The programs `XslXPathSample` and `XvmXPathSample` have sources:

`XslXPathSampleGen.hpp`, XslXPathSampleGen.cpp, `XslXPathSampleMain.cpp`, `XslXPathSampleForce.cpp`;

and `XvmXPathSampleGen.hpp`, `XvmXPathSampleGen.cpp`, `XvmXPathSampleMain.cpp`, `XvmXPathSampleForce.cpp`.

> **See Also**
>
> *Oracle AI Database XML C++ API Reference* package XPATH APIs for C++

Copyright © 1999, 2025, Oracle and/or its affiliates.

Copyright © 1999, 2025, Oracle and/or its affiliates.

32
Using the XML Class Generator for C++

Topics here explain how to use the Extensible Markup Language (XML) class generator for C++.

32.1 Accessing the XML C++ Class Generator

The XML C++ class generator is provided with Oracle Database.

32.2 Using the XML C++ Class Generator

The XML C++ class generator creates source files from an XML document type definition (DTD) or XML schema. It generates a class for each defined element. The classes are then used in a C++ program to construct XML documents that conform to the DTD or XML schema.

This is useful when an application wants to send an XML message to another application based on an agreed-upon DTD or XML Schema, 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.

The class generator works with the Oracle XML parser for C++, which parses the input and passes the parsed document to the class generator.

32.2.1 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 `-d` option that is used to parse external DTDs.

32.3 Using the XML C++ Class Generator Command-Line Utility

The standalone class generator can be called as an executable by invoking `bin/xmlcg`.

You can invoke the C++ class generator from the command line:

```
xmlcg [options] input_file
```

Table 32-1 C++ Class Generator Options

Option	Description
`-d` *`name`*	Input is an external DTD or a DTD file. Generates *`name`*`.cpp` and *`name`*`.h`.
`-o` *`directory`*	Output directory for generated files (the default is the current directory).
`-e` *`encoding`*	Default input file encoding.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 32-1 (Cont.) C++ Class Generator Options

Option	Description
`-h`	Show this usage help.
`-v`	Show the class generator version validator options.
`-s` *`name`*	Input is an XML Schema file with the given name. Generates *`name`*`.cpp` and *`name`*`.h`.

`input_file` is the name of the parsed XML document with `<!DOCTYPE>` definitions, or parsed DTD, or an XML Schema document. The XML document must have an associated DTD.

The DTD input to the XML C++ class generator is an XML document containing a DTD, or it is an external DTD. The document body itself is ignored — only the DTD is relevant, though the document must conform to the DTD.

If invalid options were used, or no input was provided, a usage message is output.

Two source files are output, a *`name`*`.h` header file and a C++ file, *`name`*`.cpp`. These are named after the DTD file.

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 these 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.

32.3.1 Input to the XML C++ Class Generator

The 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 accepts only file names for the document and associated external entities.

32.4 Using the XML C++ Class Generator Examples

The demo XML C++ class generator files are described.

Table 32-2 XML C++ Class Generator Files

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 `Makefile` on UNIX	Batch file (on Windows) or Make file (on UNIX) to generate classes and build the sample programs.
`README`	A readme file with these instructions

Copyright © 1999, 2025, Oracle and/or its affiliates.

The `make.bat` batch file (on Windows) or `Makefile` (on UNIX) does 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.

32.4.1 XML C++ Class Generator Example 1: XML — Input File to Class Generator, CG.xml

XML file `CG.xml` is presented. It is input to XML C++ class generator. It references the DTD file, `CG.dtd`.

```
<?xml version="1.0"?>
<!DOCTYPE Sample SYSTEM "CG.dtd">
  <Sample>
    <B>Be!</B>
    <D attr="value"></D>
    <E>
      <F>Formula1</F>
      <F>Formula2</F>
    </E>
  </Sample>
```

32.4.2 XML C++ Class Generator Example 2: DTD — Input File to Class Generator, CG.dtd

DTD file `CG.dtd` is presented. It is referenced by XML file `CG.xml`, which is input to 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)>
```

32.4.3 XML C++ Class Generator Example 3: CG Sample Program

Sample program CG, `CG.cpp`, is presented.

It 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.

Copyright © 1999, 2025, Oracle and/or its affiliates.

5. Writes the constructed document to `Sample.xml`.

```
////////////////////////////////////////////////////////////////////////
// NAME        CG.cpp
// DESCRIPTION Demonstration program for C++ class generator usage
////////////////////////////////////////////////////////////////////////

#ifndef ORAXMLDOM_ORACLE
# include <oraxmldom.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
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
    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) "Formula1");
    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
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

Part IV
Oracle XML Developer's Kit Reference

Reference information is presented for Oracle XML Developer's Kit (XDK).

Copyright © 1999, 2025, Oracle and/or its affiliates.

33
XSQL Pages Reference

Reference information is presented for the XSQL pages framework.

XSQL Configuration File Parameters describes settings in the XSQL configuration file. Table 33-1 lists the legal built-in actions for XSQL pages.

Table 33-1 Built-In XSQL Elements and Action Handler Classes

XSQL Action Element	Handler Class in oracle.xml.xsql.actions	Purpose
<xsql:action>	XSQLExtensionActionHandler	Invoke a user-defined action handler, implemented in Java, for executing custom logic and including custom Extensible Markup Language (XML) data in your XSQL page.
<xsql:delete-request>	XSQLDeleteRequestHandler	Delete an existing row in the database based on the posted XML document supplied in the request.
<xsql:dml>	XSQLDMLHandler	Execute a structured query language (SQL) data manipulation language (DML) statement or a Procedural Language/Structured Query Language (PL/SQL) anonymous block.
<xsql:if-param>	XSQLIfParamHandler	Conditionally include XML content or other XSQL actions.
<xsql:include-owa>	XSQLIncludeOWAHandler	Include the results of a stored procedure that uses the Oracle Web Agent (OWA) packages in the database to generate XML.
<xsql:include-param>	XSQLGetParameterHandler	Include a parameter and its value as an element in the XSQL page.
<xsql:include-posted-include-posted>	XSQLIncludePostedXMLHandler	Include the XML document that has been posted in the request into the XSQL page.
<xsql:include-request-params>	XSQLIncludeRequestHandler	Include all request parameters as XML elements in the XSQL page.
<xsql:include-xml>	XSQLIncludeXMLHandler	Include arbitrary XML resources at any point in your page by relative or absolute URL.
<xsql:include-xsql>	XSQLIncludeXSQLHandler	Include the results of one XSQL page at any point inside another.
<xsql:insert-param>	XSQLInsertParameterHandler	Insert the XML document contained in the value of a single parameter.
<xsql:insert-request>	XSQLInsertRequestHandler	Insert the XML document or HTML form posted in the request into a table or view.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 33-1 (Cont.) Built-In XSQL Elements and Action Handler Classes

XSQL Action Element	Handler Class in oracle.xml.xsql.actions	Purpose
<xsql:query>	XSQLQueryHandler	Execute an arbitrary SQL statement and include its result in canonical XML format.
<xsql:ref-cursor-function>	XSQLRefCursorFunctionHandler	Include the canonical XML representation of the result set of a cursor returned by a PL/SQL stored function.
<xsql:set-cookie>	XSQLSetCookieHandler	Set an HTTP Cookie.
<xsql:set-page-param>	XSQLSetPageParamHandler	Set an HTTP-Session level parameter. Set a page-level (local) parameter that can be referred to in subsequent SQL statements in the page.
<xsql:set-session-param>	XSQLSetSessionParamHandler	Set an HTTP-Session level parameter.
<xsql:set-stylesheet-param>	XSQLStylesheetParameterHandler	Set the value of a top-level Extensible Stylesheet Language Transformation (XSLT) parameter.
<xsql:update-request>	XSQLUpdateRequestHandler	Update an existing row in the database based on the posted XML document supplied in the request.

33.1 XSQL Configuration File Parameters

You can use the XSQL configuration file to tune your XSQL pages environment. The available configuration settings are described.

Table 33-2 XSQL Configuration File Settings

Configuration Setting Name	Description
XSQLConfig/servlet/output-buffer-size	Sets the size in bytes of the buffered output stream. If the servlet engine already buffers I/O to the servlet output stream, you can set to 0 (the default) to avoid additional buffering. Any nonnegative integer is valid.
XSQLConfig/servlet/suppress-mime-charset/media-type	The XSQL servlet sets the HTTP ContentType header to indicate the Multipurpose Internet Mail Extensions (MIME) type of the resource returned to the request. By default, the servlet includes the optional character set data in the MIME type. For a particular MIME type, you can suppress the inclusion of the character set data by including a <media-type> element, with the desired MIME type as its contents. You can list any number of <media-type> elements. Valid value is any string.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 33-2 (Cont.) XSQL Configuration File Settings

Configuration Setting Name	Description
`XSQLConfig/processor/character-set-conversion/default-charset` **Note**: Setting name is a single line. It is displayed on two lines due to space constraints.	Performs character set conversion by default 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 `8859_1`, which corresponds to the Internet Assigned Numbers Authority (IANA) `ISO-8859-1` set. If your servlet engine uses a different character set as its base, then you can specify this 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 technique is useful if you are working with parameter values that are correctly representable with your servlet default character set. It eliminates overhead associated with performing the character set conversion. Valid values are any Java character set name or `<none/>`.
`XSQLConfig/processor/reload-connections-on-error`	Connection definitions are cached when the XSQL pages processor is initialized. Set to `yes` (default) 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 is not in the cached connection list. The `yes` setting is useful for 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. 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. It takes effect only when you use the Oracle JDBC driver; otherwise the setting is ignored. This technique reduces network round trips to the database from the servlet engine running in a different tier. Default is `50`. Valid value is any nonzero positive integer.
`XSQLConfig/processor/page-cache-size`	Sets the size of the cache for XSQL page templates and so determines the maximum number of XSQL pages that are cached. Least recently used pages move out of the cache if you go above this number. Default is 25. Any nonzero positive integer is valid.
`XSQLConfig/processor/stylesheet-cache-size`	Sets the size of the cache for XSLT stylesheets and so determines the maximum number of XSQL pages that are cached. Least recently used pages move out of the cache if you go above this number. Default is 25. Any nonzero positive integer is valid.
`XSQLConfig/processor/stylesheet-pool/initial`	Each cached stylesheet is 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 nonzero positive integer.
`XSQLConfig/processor/stylesheet-pool/increment`	Sets the number of stylesheets allocated when the stylesheet pool must grow due to increased load on the server. Default is 1. Valid value is any nonzero positive integer.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 33-2 (Cont.) XSQL Configuration File Settings

Configuration Setting Name	Description
XSQLConfig/processor/stylesheet-pool/timeout-seconds	Sets the number of seconds of inactivity before a stylesheet instance in the pool is removed to free resources as the pool tries to shrink back to its initial size. Default is 60. Valid value is any nonzero positive integer.
XSQLConfig/processor/connection-pool/initial	Controls the initial number of Java Database Connectivity (JDBC) connections allocated in each connection pool. The XSQL pages processor's default connection manager implements connection pooling to improve throughput. Default is 2. Valid value is any nonzero positive integer.
XSQLConfig/processor/connection-pool/increment	Sets the number of connections allocated when the connection pool must grow due to increased load on the server. Default is 1. Valid value is any nonzero positive integer.
XSQLConfig/processor/connection-pool/timeout-seconds	Sets the number of seconds of inactivity before a JDBC connection in the pool is removed to free resources as the pool tries to shrink back to its initial size. Default is 60. Valid value is any nonzero 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, default is XSQLConnectionManagerFactoryImpl. Valid value is any class name that implements the XSQLConnectionManagerFactory interface.
XSQLConfig/processor/owa/fetch-style	Sets the default OWA Page Buffer fetch style used by the <xsql:include-owa> action. Valid values are CLOB (default) or TABLE. If set to CLOB, then the processor uses a temporary CLOB to retrieve the OWA page buffer. If set to TABLE, then the processor uses a more efficient approach that requires the Oracle Database user-defined type XSQL_OWA_ARRAY. Create this type with this data definition language (DDL) statement: CREATE TYPE xsql_owa_array AS TABLE OF VARCHAR2(32767)
XSQLConfig/processor/timing/page	Determines whether 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. Valid values are yes or no (default).
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. Valid values are yes or no (default).

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 33-2 (Cont.) XSQL Configuration File Settings

Configuration Setting Name	Description
`XSQLConfig/processor/logger/factory`	Specifies the fully qualified Java class name of a custom XSQL logger factory implementation. If not set, then no logger is used. Valid value is any class name that implements the `XSQLLoggerFactory` interface.
`XSQLConfig/processor/error-handler/class`	Specifies the fully qualified Java class name of a custom XSQL error handler. The specified handler is the default error handler implementation. If not set, then the default error handler is used. Valid value is any class name that implements the `XSQLErrorHandler` interface.
`XSQLConfig/processor/xml-parsing/preserve-whitespace`	Determines whether the XSQL pages processor preserves white space when parsing XSQL pages and XSLT stylesheets. Valid values are `true` (default) or `false`. Changing the default to false can slightly speed up processing of XSQL pages and stylesheets because ignoring white space while parsing is faster than preserving it.
`XSQLConfig/processor/security/stylesheet/defaults/allow-client-style` **Note**: Setting name is a single line. It is displayed on two lines due to space constraints.	Prevents client overriding of the stylesheet. Valid values are `yes` and `no`. During development it is sometimes useful to use the XSQL stylesheet override feature by providing a value for the `xml-stylesheet` parameter in the request. You can use the `xml-stylesheet=none` combination to temporarily disable the application of the stylesheet for debugging purposes. You can add the `allow-client-style="no"` attribute to the document element of each XSQL page to prohibit client overriding of the stylesheet in production applications. This setting can globally change the default behavior for `allow-client-style` in a single place. This setting specifies only *default* behavior. If the attribute value is explicitly specified on the document element for a given XSQL page, its value takes precedence over this global default.
`XSQLConfig/processor/security/stylesheet/trusted-hosts/host` **Note**: Setting name is a single line. It is displayed on two lines due to space constraints.	Specifies that any absolute URL to an XSLT stylesheet must be from a trusted host whose name is listed in the configuration file. List any number of `<host>` elements inside the `<trusted-hosts>` element. The name of the local machine, `localhost`, and `127.0.0.1` are trusted hosts by default. Valid values are any host name or IP address. The XSLT processor supports Java extension functions. Typically, XSQL pages refer to XSLT stylesheets with relative URLs.
`XSQLConfig/http/proxyhost`	Sets the name of the HTTP proxy server to use when processing URLs with the HTTP protocol. Valid value is any host name or Internet Protocol (IP) address.
`XSQLConfig/http/proxyport`	Sets the port number of the HTTP proxy server to use when processing URLs with the HTTP protocol. Valid value is any nonzero integer.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 33-2 (Cont.) XSQL Configuration File Settings

Configuration Setting Name	Description
XSQLConfig/connectiondefs/connection	Defines a short name and the JDBC details for a named connection used by the XSQL pages processor. You may supply any number of <connection> element children of <connectiondefs>. Each connection definition must supply a name attribute and may supply children elements <username>, <password>, <driver>, <dburl>, and <autocommit>.
XSQLConfig/connectiondefs/connection/username	Defines the user name 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 used for the current connection. If not specified, defaults to oracle.jdbc.driver.OracleDriver.
XSQLConfig/connectiondefs/connection/autocommit	Explicitly sets the Auto Commit flag for the current connection. If not specified, the connection uses the JDBC driver default setting for Auto Commit.
XSQLConfig/serializerdefs/serializer	Defines a named custom serializer implementation. You can 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 XSQLDocumentSerializer interface.

33.2.1 <xsql:action>

Element <xsql:action> is described.

Purpose

Invokes a user-defined action handler, implemented in Java, for executing custom logic and including custom XML data in a XSQL page. The Java class invoked with this action must implement the oracle.xml.xsql.XSQLActionHandler interface.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Use <xsql:action> to perform tasks that are not handled by the built-in action handlers. Custom actions can supply arbitrary XML content to the data page and perform arbitrary processing.

Usage Notes

The XSQL page processor processes the actions in a page in this way:

1. Constructs an instance of the action handler class with 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)`.

Syntax

The syntax for this action is as follows, where `handler` is a single, required attribute named whose value is the fully qualified Java class name of the invoked action, *`yourpackage`* is the Java package, and *`YourCustomHandler`* is the Java class:

```
<xsql:action handler="yourpackage.YourCustomHandler"/>
```

Some action handlers expect text content or element content to appear inside the <xsql:action> element. In this case, use syntax such as:

```
<xsql:action handler="yourpackage.YourCustomHandler">
  Some_text
</xsql:action>
```

You can also use this syntax:

```
<xsql:action handler="yourpackage.YourCustomHandler">
  <some>
    <other/>
    <elements/>
    <here/>
  </some>
</xsql:action>
```

Attributes

The only required attribute is `handler`, but you can supply additional attributes to the handler. For example, if `yourpackage.YourCustomHandler` is expecting attributes named `param1` and `param2`, then use this syntax:

```
<xsql:action handler="yourpackage.YourCustomHandler" param1="xxx" param2="yyy">
```

Examples

The following example shows an XSQL page that invokes the `myactions.StockQuotes` Java class. It includes stock quotes from Google for any symbols passed in with the `symbol` parameter. If this parameter is not supplied, it supplies a default list.

Retrieving Stock Quotes

```
<?xml version="1.0"?>
<page xmlns:xsql="urn:oracle-xsql">
  <xsql:action handler="myactions.StockQuotes"
               symbols="{@symbol}"
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
                    symbol="ORCL,SAP,MSFT,IBM"/>
</page>
```

33.2.2 <xsql:delete-request>

Element `<xsql:delete-request>` is described.

Purpose

Accepts data posted from an XML document or HTML form and uses the XML SQL Utility (XSU) to delete the content of an XML document in canonical form from a target table or view.

By combining XSU with XSLT, you can transform XML into the canonical format expected by a given table. Afterward, you can use XSU to delete the resulting canonical XML. For a specified database table, the canonical XML form is given by one row of XML output from a `SELECT *` query against the table.

Syntax

The syntax for this action is as follows, where *`table_name`* is the name of a table and *`key`* is a list of one or more columns to use as the unique key:

```
<xsql:delete-request table="table_name" key-columns="key"/>
```

Attributes

Table 33-3 lists the optional attributes that you can use on the `<xsql:delete-request>` action. Required attributes are in bold

Table 33-3 Attributes for <xsql:delete-request>

Attribute Name	Description
`table = "string"`	Name of the table, view, or synonym to use for deleting the XML data.
`key-columns = "string string ..."`	Space-delimited or comma-delimited list of one or more column names. The processor uses the values of these names in the posted XML document to identify the existing rows to delete.
`transform = "URL"`	Relative or absolute URL of the XSLT transformation to use to transform the document to be deleted into canonical ROWSET/ROW format.
`columns = "string"`	Relative or absolute URL of the XSLT transformation to use to transform the document to be deleted into canonical ROWSET/ROW format.
`commit = "boolean"`	If set to `yes` (default), invokes `COMMIT` on the current connection after a successful execution of the deletion. Valid values are `yes` and `no`.
`commit-batch-size = "integer"`	If a positive, nonzero `integer` is specified, then after each batch of `integer` deleted records, the processor issues a `COMMIT`. The default batch size is zero (0) if not specified, which means that the processor does not commit interim batches.
`date-format = "string"`	Date format mask to use for interpreting date field values in XML being deleted. Valid values are those documented for the `java.text.SimpleDateFormat` class.
`error-param = "string"`	Name of a page-private parameter that must be set to the string `Error` if a nonfatal error occurs while processing this action. Valid value is any parameter name.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Examples

The following example specifies that the posted XML document is to be transformed with the `style.xsl` stylesheet and then deleted from the `departments` table. The `departments.department_id` column is the primary key for the deletion.

Deleting Rows

```
<?xml version="1.0"?>
<xsql:delete-request table="departments"       transform="style.xsl"
 connection="demo" key-columns="department_id" xmlns:xsql="urn:oracle-xsql"/>
```

33.2.3 <xsql:dml>

Element `<xsql:dml>` is described.

Purpose

Executes a DML or DDL statement or a PL/SQL block. Typically, you use this tag to include statements that would be executed or rolled back together.

This action requires a database connection provided as a `connection="connname"` attribute on the document element of the XSQL page in which it appears.

Usage Notes

You cannot set parameter values by binding them in the position of `OUT` variables with `<xsql:dml>`. Only `IN` parameters are supported for binding.

Syntax

The syntax for the action is as follows, where *`DML_DDL_or_PLSQL`* is a placeholder for a legal DML statement, DDL statement, or PL/SQL block:

```
<xsql:dml>
  DML_DDL_or_PLSQL
</xsql:dml>
```

Attributes

Table 33-4 lists the optional attributes that you can use on the `<xsql:dml>` action.

Table 33-4 Attributes for <xsql:dml>

Attribute Name	Description
`commit = "boolean"`	If set to `yes`, invokes commit on the current connection after a successful execution of the DML statement. Valid values are `yes` and `no` (default).
`bind-params = "string"`	Ordered, space-delimited list of one or more XSQL parameter names. The values of these parameters are used to bind to the JDBC bind variable in the appropriate sequential position in the SQL statement.
`error-param = "string"`	Name of a page-private parameter that must be set to the string '`Error`' if a nonfatal error occurs while processing this action. Valid value is any parameter name.
`error-statement = "boolean"`	If set to `no`, suppresses the inclusion of the offending SQL statement in any `<xsql-error>` element generated. Valid values are `yes` (default) and `no`.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Examples

The following example inserts the user name stored in the `webuser` cookie into a `request_log` table. Using bind variables guards against SQL injection attacks.

Inserting a User Name into a Table

```
<xsql:dml connection="demo" bind-params="webuser"
          xmlns:xsql="urn:oracle-xsql">
  BEGIN
    INSERT INTO request_log(page,userid)
      VALUES( 'somepage.xsql', ? );
    COMMIT;
  END;
</xsql:dml>
```

33.2.4 <xsql:if-param>

Element `<xsql:if-param>` is described.

Purpose

Enables you to include elements and actions nested inside if a specified condition is true. If the condition is true, then all nested XML content and actions are included in the page. If the condition is false, then none of the nested XML content or actions is included (and thus none of the nested actions is executed).

Specify which parameter value is evaluated by supplying the required `name` attribute. Simple parameter names and array-parameter names are supported.

> **Note**
>
> If the parameter being tested does not exist, the test evaluates to false.

Syntax

The syntax for the action is this, where *`some_name`* is the value of the `name` attribute and *`test_condition`* is exactly one of the conditions listed in Table 33-5:

```
<xsql:if-param name="some_name" test_condition>
   element_or_action
</xsql:if-param>
```

Any XML content or XSQL action elements can be nested inside an `<xsql:if-param>`, including other `<xsql:if-param>` elements.

Attributes

In addition to the required `name` attribute, you must choose exactly one of the attributes listed in Table 33-5 to indicate how the parameter value (or values, in the array case) is tested. As with other XSQL actions, the attributes of the `<xsql:if-param>` action can contain lexical substitution parameter expressions such as `{@paramName}`.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 33-5 Attributes for <xsql:if-param>

Attribute Name	Description
`exists="yes_or_no"`	If set to `exists="yes"`, then this condition tests whether the named parameter exists and has a nonempty value. For an array-valued parameter, it tests whether the array-parameter exists and has at least one nonempty element. If set to `exists="no"`, then this condition evaluates to true if the parameter does not exist, of if it exists but has an empty value. For an array-valued parameter, it evaluates to true if the parameter does not exist, or if all of the array elements are empty.
`equals="stringValue"`	This condition tests whether the named parameter equals the string value provided. By default the comparison is an exact string match. For a case-insensitive match, supply the additional `ignore-case="yes"` attribute as well. For an array-valued parameter, the condition tests whether any element in the array has the indicated value.
`not-equals="stringValue"`	This condition tests whether the named parameter does not equal the string value provided. By default the comparison is an exact string match. For an array-valued parameter, the condition evaluates to true if none of the elements in the array has the indicated value.
`in-list = "comma-or-space-separated-list"`	This condition tests whether the named parameter matches any of the strings in the provided list. By default the comparison is an exact string match. For a case-insensitive match, supply the additional `ignore-case="yes"` attribute as well. The value of the `in-list` parameter is tokenized into an array with commas as the delimiter if commas are detected in the string. Otherwise, it uses a space as the delimiter. For an array-valued parameter, the condition tests whether any element in the array matches an element in the list.
`not-in-list = "comma-or-space-separated-list"`	This tests whether the named parameter does not match any of the strings in the provided list. By default the comparison is an exact string match. For a case-insensitive match, supply the additional `ignore-case="yes"` attribute as well. The value of the `not-in-list` parameter is tokenized into an array with commas as the delimiter if commas are in the string. Otherwise, the processor uses a space as the delimiter. For an array-valued parameter, the condition tests whether none of the elements in the array matches an element in the list.

Examples

To test whether two different conditions are true, you can use nested `<xsql:if-param>` elements as shown in the following example.

Testing Conditions

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="style.xsl"?>
<page connection="demo" xmlns:xsql="urn:oracle-xsql">
<!--
| Set page parameter 'some_param' to value "some_value" if parameter 'a'
| exists, and if parameter 'b' has a value equal to "X"
+-->
  <xsql:if-param name="a" exists="yes">
    <xsql:if-param name="b" equals="X">
      <xsql:set-page-param name="some_param" value="some_value"/>
    </xsql:if-param>
  </xsql:if-param>
  <!-- ... -->
</page>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

33.2.5 <xsql:include-owa>

Element <xsql:include-owa> is described.

Purpose

Includes XML content 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 OWA packages (HTP and HTF) to "print" the XML tags into the server-side page buffer. Afterwards, the XSQL pages 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.

Usage Notes

You can create a wrapper procedure that constructs XML elements with the HTP package. Your XSQL page can invoke the wrapper procedure by using <xsql:include-owa>.

Syntax

The syntax for the action is as follows, where *PL/SQL_block* is a PL/SQL Block invoking a procedure that uses the HTP or HTF packages:

```
<xsql:include-owa>
   PL/SQL_block
</xsql:include-owa>
```

Attributes

Table 33-6 lists the optional attributes supported by this action.

Table 33-6 Attributes for <xsql:include-owa>

Attribute Name	Description
bind-params = "*string*"	Ordered, space-delimited list of one or more XSQL parameter names. The values of these parameters are used to bind to the JDBC bind variable in the appropriate sequential position in the SQL statement.
error-param = "*string*"	Name of a page-private parameter that must be set to the string 'Error' if a nonfatal error occurs while processing this action. Valid value is any parameter name.
error-statement = "*boolean*"	If set to no, suppresses the inclusion of the offending SQL statement in any <xsql-error> element generated. Valid values are yes (default) and no.

Examples

Assume that you write a PL/SQL procedure called UpdateStatus that updates the status of a project. The procedure uses HTP to print an <UpdateStatus> datagram that contains the element <Success/> if no errors occur or one or more <Error> elements if errors occur.

The following example shows how you can invoke UpdateStatus from an XSQL page. The example uses SQL bind variable instead of lexical substitution to prevent the possibility of SQL injection attacks.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Including XML Content Created by a Stored Procedure

```
<xsql:include-owa connection="demo"
                  bind-params="project status"
                  xmlns:xsql="urn:oracle-xsql">
  UpdateStatus( ?,? );
</xsql:include-owa>
```

Assume that a user enters an invalid status number for a project into a web-based form. The form posts the input parameters to an XSQL page as shown in the following example. The XSQL processor returns this datagram, which an XSLT stylesheet could transform into an HTML error page:

```
<UpdateStatus>
  <Error Field="status">Status must be 1, 2, 3, or 4</Error>
</UpdateStatus>
```

33.2.6 <xsql:include-param>

Element `<xsql:include-param>` is described.

Purpose

Includes an XML representation of the name and value of a single parameter. This technique is useful if an associated XSLT stylesheet must refer to parameter values with XPath expressions.

Syntax

The syntax of the action is as follows, where *`paramname`* is the name of a parameter:

```
<xsql:include-param name="paramname" />
```

The required `name` attribute supplies the name of the parameter whose value you want to include.

Attributes

The `name` attribute is required; there are no optional attributes.

Examples

The following example uses XPATH to get the value of a parameter and represent it in XML.

Including an XML Representation of a Parameter Value

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="style.xsl"?>
<page connection="demo" xmlns:xsql="urn:oracle-xsql"
                        xmlns:p="http://www.companysite.com/products">
  <xsql:set-page-param name="productid"
                       xpath="/p:Products/productid"/>
  <xsql:include-param name="productid"/>
</page>
```

The XML fragment included in the datagram is:

```
<productid>12345</productid>
```

You can use an array parameter name to indicate that the value is to be treated as an array, as shown in this example:

Copyright © 1999, 2025, Oracle and/or its affiliates.

```
<xsql:include-param name="productid[]"/>
```

The XML fragment reflects all of the array values, as shown in this example:

```
<productid>
  <value>12345<value>
  <value>33455</value>
  <value>88199</value>
</productid>
```

In this array-parameter name scenario, if `productid` is a single-valued parameter, then the fragment looks identical to a one-element array, as showd in this example:

```
<productid>
  <value>12345<value>
</productid>
```

33.2.7 <xsql:include-posted-include-posted>

Element `<xsql:dml>` is described.

Purpose

Includes the posted XML document in the XSQL page. If the user posts an HTML form instead of an XML document, then the XML included is similar to that included by the `<xsql:include-request-params>` action.

Syntax

The syntax of the action is:

```
<xsql:include-posted-xml/>
```

Attributes

None.

Examples

The following example shows a sample XSQL page that includes a posted XML document.

Including Posted XML

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsql" href="somepage.xsql"?>
<page connection="demo"
      xmlns:xsql="urn:oracle-xsql">
  <xsql:include-posted-xml/>
</page>
```

33.2.8 <xsql:include-request-params>

Element `<xsql:include-request-params>` is described.

Purpose

Includes an XML representation of all parameters in the request in the datagram. The action element is replaced in the page at page-request time with a tree of XML elements that represents the parameters available to the request.

Copyright © 1999, 2025, Oracle and/or its affiliates.

This technique is useful if an associated XSLT stylesheet must refer to request parameter values with XPath expressions.

Usage Notes

When processing pages through the XSQL servlet, the XML included takes the form shown in the following example.

Including Request Parameters

```
<request>
  <parameters>
    <paramname>value1</paramname>
    <ParamName2>value2</ParamName2>
     ...
  </parameters>
  <session>
    <sessVarName>value1</sessVarName>
     ...
  </session>
  <cookies>
    <cookieName>value1</cookieName>
     ...
  </cookies>
</request>
```

When you use the XSQL command-line utility or the `XSQLRequest` class, the XML takes the form shown in the following example.

Including Request Parameters

```
<request>
  <parameters>
    <paramname>value1</paramname>
    <ParamName2>value2</ParamName2>
     ...
  </parameters>
</request>
```

The technique enables you to distinguish request parameters from session parameters or cookies because its value is a child element of `<parameters>`, `<session>`, or `<cookies>`.

Syntax

The syntax of the action is:

```
<xsql:include-request-params/>
```

Attributes

None.

Examples

The following example shows a sample XSQL page that includes all request parameters in the data page.

Including Request Parameters

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsql" href="cookie_condition.xsl"?>
<page connection="demo" xmlns:xsql="urn:oracle-xsql">
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
  <xsql:include-request-params/>
</page>
```

The `cookie_condition.xsl` stylesheet chooses an output format based on whether the `siteuser` cookie is present. The following example shows a fragment of the stylesheet.

Testing for Conditions in a Stylesheet

```
<xsl:choose>
  <xsl:when test="/page/request/cookies/siteuser">
  ...
  </xsl:when>
  <xsl:otherwise>
  ...
  </xsl:otherwise>
</xsl:choose>
```

33.2.9 <xsql:include-xml>

Element `<xsql:include-xml>` is described.

Purpose

Includes the XML contents of a local, remote, or database-driven XML resource in your datagram. You can specify the resource by URL or SQL statement. The server can deliver a resource that is a static XML file or dynamically created XML from a programmatic resource such as a servlet or common gateway interface (CGI) program.

Syntax

The syntax for this action is as follows, where *`URL`* is a relative URL or an absolute, HTTP-based URL to retrieve XML from another web site:

```
<xsql:include-xml href="URL"/>
```

Alternatively, you can use this syntax, where *`SQL_statement`* is a SQL `SELECT` statement selecting a single row containing a single `CLOB` or `VARCHAR2` column value:

```
<xsql:include-xml>
  SQL_statement
</xsql:include-xml>
```

The `href` attribute and SQL statement are mutually exclusive. If you provide one, then the other is not allowed.

Attributes

Table 33-7 lists the attributes supported by this action. Required attributes are in bold.

Table 33-7 Attributes for <xsql:include-xml>

Attribute Name	Description
`href="URL"`	The absolute, relative, or parameterized URL of the XML resource to be included. The resource can be a static file dynamic source.
`bind-params = "string"`	Ordered, space-delimited list of one or more XSQL parameter names. The values for these names are used to bind to the JDBC bind variable in the appropriate sequential position in the SQL statement.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 33-7 (Cont.) Attributes for <xsql:include-xml>

Attribute Name	Description
`error-param = "string"`	Name of a page-private parameter that must be set to the string 'Error' if a nonfatal error occurs while processing this action. Valid value is any parameter name.

Examples

The following example includes an XML document retrieved by a database query. The XML content is a CLOB-valued member field of a user-defined type. The XML included must come from a VARCHAR2 or CLOB column, not an XMLType.

Including an XML Document

```
<?xml version="1.0"?>
<xsql:include-xml bind-params="id" connection="demo"
                  xmlns:xsql="urn:oracle-xsql">
  SELECT x.document.contents doc FROM xmldoc x
  WHERE x.docid = ?
</xsql:include-xml>
```

33.2.10 <xsql:include-xsql>

Element <xsql:include-xsql> is described.

Purpose

Includes the XML output of one XSQL page in another page. You can create a page that assembles the contents—optionally transformed—from other XSQL pages.

Usage Notes

If the aggregated page contains an <?xml-stylesheet?> processing instruction, then this stylesheet is applied before the result is aggregated. Thus, you can use <xsql:include-xsql> to chain XSLT stylesheets.

When one XSQL page aggregates another page by using <xsql:include-xsql>, all request-level parameters are visible to the nested page. For pages processed by the XSQL Servlet, the visible data includes session-level parameters and cookies. None of the page-private parameters of the aggregating page are visible to the nested page.

Syntax

The syntax for this action is as follows, where *XSQL_page* is a relative or absolute URL of an XSQL page to be included:

```
<xsql:include-xsql href="XSQL_page"/>
```

Attributes

Table 33-8 lists the attributes supported by this action. Required attributes are in bold; all others are optional.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 33-8 Attributes for <xsql:include-xsql>

Attribute Name	Description
href="*string*"	Relative or absolute URL of XSQL page to be included.
error-param = "*string*"	Name of a page-private parameter that must be set to the string `Error` if a nonfatal error occurs while processing this action. Valid value is any parameter name.
reparse = "*boolean*"	Indicates whether output of the included XSQL page must be reparsed before it is included. Valid values are `no` (default) and `yes`. This attribute is useful if the included XSQL page selects the text of an XML document fragment that the including page wants to treat as elements.

Examples

The following example displays an XSQL page that lists discussion forum categories.

Categories.xsql

```
<?xml version="1.0"?>
<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
  SELECT name
  FROM categories
  ORDER BY name
</xsql:query>
```

The following example shows how you can include the results of the page in the previous Categories.xsql example into a page that lists the ten most recent topics in the current forum.

TopTenTopics.xsql

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="style.xsl"?>
<top-ten-topics connection="demo" 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 also use `<xsql:include-xsql>` to apply an XSLT stylesheet to an included page. Assume that you write this XSLT stylesheets:

- `cats-as-html.xsl`, which renders the topics in HTML
- `cats-as-wml.xsl`, which renders the topics in WML

One approach for catering to two different types of devices is to create different XSQL pages for each device. The following example shows an XSQL page that aggregates `Categories.xsql` and applies the `cats-as-html.xsl` stylesheet.

HTMLCategories.xsql

Copyright © 1999, 2025, Oracle and/or its affiliates.

```
<?xml version="1.0"?>
<!-- HTMLCategories.xsql -->
<?xml-stylesheet type="text/xsl" href="cats-as-html.xsl"?>
<xsql:include-xsql href="Categories.xsql" xmlns:xsql="urn:oracle-xsql"/>
```

The following example shows an XSQL page that aggregates `Categories.xsql` and applies the `cats-as-html.xsl` stylesheet for delivering to wireless devices.

WMLCategories.xsql

```
<?xml version="1.0"?>
<!-- WMLCategories.xsql -->
<?xml-stylesheet type="text/xsl" href="cats-as-wml.xsl"?>
<xsql:include-xsql href="Categories.xsql" xmlns:xsql="urn:oracle-xsql"/>
```

33.2.11 <xsql:insert-param>

Element `<xsql:insert-param>` is described.

Purpose

Inserts the value of a parameter into a table or view. Use this tag when the client is posting a well-formed XML document as text in an HTTP parameter or individual HTML form field.

By combining the XML SQL Utility (XSU) with XSLT, you can transform XML into the canonical format expected by a given table. Afterward, you can use XSU to insert the resulting canonical XML. For a specified database table, the canonical XML form is given by one row of XML output from a `SELECT *` query against the table.

Syntax

The syntax for this action is as follows, where *`table_or_view_name`* is a relative or absolute URL of an XSQL page to be included:

```
<xsql:insert-param table="table_or_view_name" name="string"/>
```

Attributes

Table 33-9 lists the optional attributes that you can use on the `<xsql:insert-param>` action.

Table 33-9 Attributes for <xsql:insert-param>

Attribute Name	Description
`name="string"`	Name of the parameter whose value contains XML to be inserted.
`table="string"`	Name of the table, view, or synonym to use for inserting the XML data.
`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-delimited or comma-delimited list of one or more column names whose values are inserted. If supplied, then only these columns are inserted. If not supplied, all columns are inserted, with NULL values for columns whose values do not appear in the XML document.
`commit = "boolean"`	If set to `yes`, invokes commit on the current connection after a successful execution of the insert. Valid values are `yes` (default) and `no`.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 33-9 (Cont.) Attributes for <xsql:insert-param>

Attribute Name	Description
`commit-batch-size = "integer"`	If a positive, nonzero number *integer* is specified, then after each batch of *integer* inserted records, the XSQL processor issues a `COMMIT`. Default batch size is zero (`0`), which instructs the processor 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 for the `java.text.SimpleDateFormat` class.
`error-param = "string"`	Name of a page-private parameter that must be set to `Error` if a nonfatal error occurs while processing this action. Valid value is any parameter name.

Examples

The following example parses and transforms the contents of the HTML form parameter `xmlfield` for database insert.

Inserting XML Contained in an HTML Form Parameter

```
<?xml version="1.0"?>
<xsql:insert-param name="xmlfield" table="image_metadata_table"
transform="field-to-rowset.xsl" connection="demo" xmlns:xsql="urn:oracle-xsql"/>
```

33.2.12 <xsql:insert-request>

Element `<xsql:insert-request>` is described.

Purpose

Accepts data posted from an XML document or HTML form and uses the XML SQL Utility (XSU) to insert the content of an XML document in canonical form into a target table or view.

If an HTML Form has been posted, then the posted XML document is materialized from HTTP request parameters, cookies, and session variables. The XML document has this form:

```
<request>
<parameters>
  <param1>value1</param1>
    :
  </paramN>valueN</paramN>
</parameters>
  :
</request>
```

By combining XSU with XSLT, you can transform XML into the canonical format expected by a given table. The XSQL engine uses XSU to insert the resulting canonical XML. For a specified database table, the canonical XML form is given by one row of XML output from a `SELECT *` query against the table.

Usage Notes

If you target a database view with an `INSERT`, then you can create `INSTEAD OF INSERT` triggers on the view to further automate the handling of the posted data. For example, an `INSTEAD OF INSERT` trigger on a view can 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.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Syntax

The syntax for this action is:

```
<xsql:insert-request table="table"/>
```

Attributes

Table 33-10 lists the optional attributes that you can use on the `<xsql:insert-request>` action.

Table 33-10 Attributes for <xsql:insert-request>

Attribute Name	Description
`table = "string"`	Name of the table, view, or synonym to use for inserting the XML data.
`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"`	Relative or absolute URL of the XSLT transformation to use to transform the document to be inserted into canonical ROWSET/ROW format.
`commit = "boolean"`	If set to `yes` (default), invokes `COMMIT` on the current connection after a successful execution of the insert. Valid values are `yes` and `no`.
`commit-batch-size = "integer"`	If a positive, nonzero number `integer` is specified, then after each batch of integer inserted records, the processor issues a `COMMIT`. The default batch size is zero (0) if not specified, which means that the processor does not 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 `java.text.SimpleDateFormat` class.
`error-param = "string"`	Name of a page-private parameter that must be set to the string `Error` if a nonfatal error occurs while processing this action. Valid value is any parameter name.

Examples

The following example parses and transforms the contents of the posted XML document or HTML Form for insert.

Inserting XML Received in a Parameter

```
<?xml version="1.0"?>
<xsql:insert-request
  table="purchase_order"
  transform="purchseorder-to-rowset.xsl"
  connection="demo"
  xmlns:xsql="urn:oracle-xsql"/>
```

33.2.13 <xsql:query>

Element `<xsql:query>` is described.

Purpose

Executes a SQL select statement and includes a canonical XML representation of the query 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.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Syntax

The syntax for the action is:

```
<xsql:query>
   SELECT_Statement
</xsql:query>
```

Any legal SQL select statement is permissible as a substitution for the *SELECT_Statement* placeholder. If the select statement produces no rows, then you can provide a fallback query by including a nested <xsql:no-rows-query> element:

```
<xsql:query>
  SELECT_Statement
  <xsql:no-rows-query>
    Fallback_SELECT_Statement
  </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 legal on the <xsql:query> action element.

Attributes

The optional attributes listed in Table 33-11 can be supplied to control various aspects of the data retrieved and the XML produced by the <xsql:query> action.

Table 33-11 Attributes for <xsql:query>

Attribute Name	Description
bind-params = "*string*"	Ordered, space-delimited list of one or more XSQL parameter names. The values of these parameters are used to bind to the JDBC bind variable in the appropriate sequential position in the SQL statement.
date-format = "*string*"	Date format mask to use for formatted date column and attribute values in the XML that is queried. Valid values are the same values legal for the java.text.SimpleDateFormat class.
error-param = "*string*"	Name of a page-private parameter that must be set to the string 'Error' if a nonfatal error occurs while processing this action. Valid value is any parameter name.
error-statement = "*boolean*"	If set to no, suppresses the inclusion of the offending SQL statement in any <xsql-error> element generated. Valid values are yes (default) and no.
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 /XSQLConfig/processor/default-fetch-size configuration setting in XSQLConfig.xml.
id-attribute = "*string*"	XML attribute name to use instead of the default num for uniquely identifying each row in the result set. If the value is the empty string, then the row id attribute is suppressed.
id-attribute-column = "*string*"	Case-sensitive name of the column in the result set whose value must 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 yes, includes an inline XML schema that describes the structure of the result set. Valid values are yes and no (default).

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 33-11 (Cont.) Attributes for <xsql:query>

Attribute Name	Description
`max-rows = "integer"`	Maximum number of rows to fetch after optionally skipping the number of rows set by the `skip-rows` attribute. If not specified, the default is to fetch all rows.
`null-indicator = "boolean"`	Indicates whether to signal that a column's value is NULL by including the `NULL="Y"` attribute on the element for the column. By default, columns with NULL values are omitted from the output. Valid values are `yes` and `no` (default).
`row-element = "string"`	XML element name to use instead of the default `<ROW>` for the `rowset` of query results. Set to the empty string to suppress generating a containing `<ROW>` element for each row in the result set.
`rowset-element = "string"`	XML element name to use instead of the default `<ROWSET>` for the `rowset` of query results. Set to the empty string to suppress generating a containing `<ROWSET>` element.
`skip-rows = "integer"`	Number of rows to skip before fetching rows from the result set. Can be combined with `max-rows` for stateless paging through query results.
`tag-case = "string"`	Valid values are `lower` and `upper`. If not specified, the default is to use the case of column names as specified in the query as corresponding XML element names.

Examples

The following example shows a simple XSQL page.

Hello World

```
<?xml version="1.0"?>
<xsql:query connection="xmlbook" xmlns:xsql="urn:oracle-xsql">
   SELECT 'Hello, World!' AS text     FROM DUAL</xsql:query>
```

If you save the previous example as `hello.xsql` and execute it in a browser, the XSQL page processor returns this XML:

```
<?xml version = '1.0'?>
<ROWSET>
   <ROW num="1">
      <TEXT>Hello, World!</TEXT>
   </ROW>
</ROWSET>
```

By default, the XML produced by a query reflects the column structure of its result set, with element names matching the names of the columns. Columns in the result with this nested structure produce nested elements that reflect this structure:

- Object types
- Collection types
- CURSOR expressions

The result of a typical query containing different types of columns and returning one row might look like the following example.

Nested Structure Example

```
<ROWSET>
  <ROW id="1">
    <VARCHARCOL>Value</VARCHARCOL>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
    <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>
    <CURSORCOL>
      <CURSORCOL_ROW>
        <COL1>Value1</COL1>
        <COL2>Value2</COL2>
      </CURSORCOR_ROW>
    </CURSORCOL>
  </ROW>
</ROWSET>
```

A <ROW> element repeats for each row in the result set. Your query can use standard SQL column aliasing to rename the columns in the result, which effectively renames the XML elements that are produced. Column aliasing is *required* for columns whose names otherwise are illegal names for an XML element.

For example, an <xsql:query> action as shown in the following example produces an error because the default column name for the calculated expression is an illegal XML element name.

Query with Error

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="style.xsl"?>
<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
  SELECT TO_CHAR(hire_date,'DD-MON')
  FROM   employees
</xsql:query>
```

You can fix the problem by using column aliasing as shown in the following example.

Query with Column Aliasing

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="style.xsl"?>
<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
  SELECT TO_CHAR(hire_date,'DD-MON') AS hiredate FROM   employees
</xsql:query>
```

33.2.14 <xsql:ref-cursor-function>

Element <xsql:ref-cursor-function> is described.

Purpose

Executes an arbitrary stored function returning a REF CURSOR and includes the query result set in canonical XML format. This action requires a database connection to be provided by

Copyright © 1999, 2025, Oracle and/or its affiliates.

supplying a connection="*connname*" attribute on the document element of the XSQL page in which it appears.

Use this tag to invoke a stored procedure that determines what the query is and returns a cursor to the query. Used in this way, this tag also provides a weak level of security because it can hide the query from direct inspection.

Syntax

The syntax of the action is as follows, where SCHEMA_NAME represents an optional database schema name, PACKAGE_NAME represents an optional PL/SQL package name, and FUNCTION_NAME (required) specifies the name of a PL/SQL function:

```
<xsql:ref-cursor-function>
  [SCHEMA_NAME.][PACKAGE_NAME.]FUNCTION_NAME(args);
</xsql:ref-cursor-function>
```

Attributes

The optional attributes are the same as for the <xsql:query> action listed in Table 33-11 except that fetch-size is not available for <xsql:ref-cursor-function>.

Examples

By exploiting dynamic SQL in PL/SQL, a function can conditionally construct a dynamic query before a cursor handle to its result set is returned to the XSQL page processor. The return value of the function must be of type REF CURSOR. Consider the PL/SQL package shown in the following example.

DynCursor PL/SQL Package

```
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
    IF id = 1 THEN -- Conditionally return a dynamic query as a REF CURSOR
      OPEN the_cursor  -- An employees Query
       FOR 'SELECT employee_id, email FROM employees';
    ELSE
      OPEN the_cursor  -- A departments Query
        FOR 'SELECT department_name, department_id FROM departments';
   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 as shown in the following example.

Executing a REF CURSOR Function

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="style.xsl"?>
<xsql:ref-cursor-function connection="demo" xmlns:xsql="urn:oracle-xsql">
  DynCursor.DynamicQuery(1);
</xsql:ref-cursor-function>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

33.2.15 <xsql:set-cookie>

Element <xsql:set-cookie> is described.

Purpose

Sets an HTTP cookie to a value. By default, the value remains for the lifetime of the current browser, but you can change its lifetime by supplying the optional max-age attribute. The value to be assigned to the cookie can be supplied by a combination of static text and other parameter values, or from the result of a SQL SELECT statement.

Because this feature is specific to the HTTP protocol, this action is effective only if the XSQL page in which it appears is processed by the XSQL servlet. If this action is encountered in an XSQL page processed by the XSQL command-line utility or the XSQLRequest programmatic application programming interface (API), then it does nothing.

Usage Notes

If you use the SQL statement option, then 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.

If you must set several cookie values based on the results of a single SQL statement, then do not use the name attribute. Instead, you can use the names attribute and supply a space-or-comma-delimited list of one or more cookie names.

Syntax

The syntax for this action is as follows, where *paramname* is the name of a parameter:

```
<xsql:set-cookie name="paramname" value="value"/>
```

Alternatively, you can use this syntax, where *SQL_statement* is a SQL SELECT statement and *paramname* is the name of a parameter:

```
<xsql:set-cookie name="paramname">
  SQL_statement
</xsql:set-cookie>
```

Either the name or the names attribute is required. The value attribute and the contained SQL statement are mutually exclusive. The number of columns in the select list must match the number of cookies being set or an error message results.

Attributes

Table 33-12 lists the attributes supported by this action. Attributes in bold are required; all others are optional.

Table 33-12 Attributes for <xsql:set-cookie>

Attribute Name	Description
name = "*string*"	Name of the cookie whose value you want to set. You must use name or names but not both.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 33-12 (Cont.) Attributes for <xsql:set-cookie>

Attribute Name	Description
names = "*string string ...*"	Space-or-comma-delimited list of the cookie names whose values you want to set. You must use `name` or `names` but not both.
`bind-params = "string"`	Ordered, space-delimited list of one or more XSQL parameter names. Values are used to bind to the JDBC bind variable in the appropriate sequential position in the SQL statement.
`domain = "string"`	Domain in which cookie value is valid and readable. If `domain` is not set explicitly, it defaults to the fully qualified host name (for example, `server.biz.com`) of the document creating the cookie.
`error-param = "string"`	Name of a page-private parameter that is set to the string '`Error`' if a nonfatal error occurs while processing this action. Valid value is any parameter name.
`ignore-empty-value = "boolean"`	Indicates whether the cookie assignment is ignored if the value to which it is being assigned is an empty string.Valid values are `yes` and `no` (default).
`immediate = "boolean"`	Indicates whether the cookie assignment is immediately visible to the current page. Typically, cookies set in the current request are not visible until the browser sends them back to the server in a subsequent request.Valid values are `yes` and `no` (default).
`max-age = "integer"`	Sets the maximum age of the cookie in *seconds*. Default is to set the cookie to expire when users current browser session terminates.
`only-if-unset = "boolean"`	Indicates whether the cookie assignment occurs only when the cookie currently does not exists.Valid values are `yes` and `no` (default).
`path = "string"`	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.
`value = "string"`	Sets the value to assign to the cookie.

Examples

The following example sets the HTTP cookie to the value of the parameter named `choice`.

Setting a Cookie to a Parameter Value

```
<?xml version="1.0"?>
<xsql:set-cookie name="last_selection"
                 value="{@choice}" xmlns:xsql="urn:oracle-xsql"/>
```

Table 33-5 sets the HTTP cookie to a value selected from the database.

Setting a Cookie to a Database-Generated Value

```
<?xml version="1.0"?>
<xsql:set-cookie name="shopping_cart_id" bind-params="user"
                 connection="demo"       xmlns:xsql="urn:oracle-xsql">
 SELECT cartmgr.new_cart_id(UPPER(?)) FROM DUAL
</xsql:set-cookie>
```

Table 33-6 sets three cookies based on the result of a single `SELECT` statement.

Setting Three Cookies

Copyright © 1999, 2025, Oracle and/or its affiliates.

```
<?xml version="1.0"?>
<xsql:set-cookie names="paramname1 paramname2 paramname3"
                 connection="demo" xmlns:xsql="urn:oracle-xsql">
  SELECT expression_or_column1, expression_or_column2, expression_or_column3
  FROM table
  WHERE clause_identifying_a_single_row
</xsql:set-cookie>
```

33.2.16 <xsql:set-page-param>

Element `<xsql:set-page-param>` is described.

Purpose

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.

Usage Notes

If you use the SQL statement option, then the program fetches a single row from the result set and assigns the parameter the value of the first column. This usage 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.

As an alternative to providing the `value` attribute, or a SQL statement, you can supply the `xpath` attribute to set the page-level parameter to the value of an XPath expression. The XPath expression is evaluated against an XML document or HTML form that has been posted to the XSQL pages processor. The value of the `xpath` attribute can be any valid XPath expression, optionally built using XSQL parameters as part of the attribute value like any other XSQL action element.

After a page-private parameter is set, subsequent action handlers can use this value as a lexical parameter, for example `{@po_id}`. Alternatively, action handlers can use this value as a SQL bind parameter value; they can reference its name in the `bind-params` attribute of any action handler that supports SQL operations.

If you must set multiple session parameter values based on the results of a single SQL statement, instead of using the `name` attribute, then you can use the `names` attribute. You can supply a list, delimited by spaces or commas, of one or more session parameter names.

Syntax

The syntax for this action is as follows, where *`paramname`* is the name of a parameter and *`value`* is a value:

```
<xsql:set-page-param name="paramname" value="value"/>
```

Alternatively, you can use this syntax, where *`SQL_statement`* is a SQL `SELECT` statement and *`paramname`* is the name of a parameter:

```
<xsql:set-page-param nname="paramname">
  SQL_statement
</xsql:set-page-param>
```

Alternatively, you can use this syntax, where *`paramname`* is the name of a parameter and where *`expression`* is an XPath expression:

```
<xsql:set-page-param name="paramname" xpath="expression"/>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

Either the `name` or the `names` attribute is required. The `value` attribute and the contained SQL statement are mutually exclusive.

Attributes

Table 33-13 lists the attributes supported by this action. Attributes in bold are required; all others are optional.

Table 33-13 Attributes for <xsql:set-page-param>

Attribute Name	Description
`name = "string"`	Name of the page-private parameter whose value you want to set.
`names = "string string ..."`	Space-or-comma-delimited list of the page parameter names whose values you want to set. Either use the `name` or the `names` attribute, but not both.
`bind-params = "string"`	Ordered, space-delimited list of one or more XSQL parameter names. The values of these parameters are used to bind to the JDBC bind variable in the appropriate sequential position in the SQL statement.
`error-param = "string"`	Name of a page-private parameter that must be set to the string '`Error`' if a nonfatal error occurs while processing this action. Valid value is any parameter name.
`ignore-empty-value = "boolean"`	Indicates whether the page-level parameter assignment is ignored if the value to which it is being assigned is an empty string.Valid values are `yes` and `no` (default).
`quote-array-values = "boolean"`	If the parameter name is a simple-valued parameter name (for example, `myparam`) and if `treat-list-as-array="yes"` is specified, then specifying `quote-array-values="yes"` surrounds each string token with single quotation marks before separating the values with commas. Valid values are `yes` and `no` (default).
`treat-list-as-array = "boolean"`	Indicates whether the string-value assigned to the parameter is tokenized into an array of separate values before assignment. If any comma is present in the string, then the comma is used for separating tokens. Otherwise, spaces are used.Valid values are `yes` and `no`. The default value is `yes` if the parameter name being set is an array parameter name (for example, `myparam[]`), and default is `no` if the parameter name being set is a simple-valued parameter name like `myparam`.
`value = "string"`	Sets the value to assign to the parameter.
`xpath = "XPathExpression"`	Sets the value of the parameter to an XPath expression evaluated against an XML document or HTML form that has been posted to the XSQL pages processor.

Examples

The following example sets multiple parameter values based on the results of a single SQL statement.

Setting Multiple Page Parameters

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="style.xsl"?>
<xsql:set-page-param names="paramname1 paramname2 paramname3"
                     connection="demo" xmlns:xsql="urn:oracle-xsql>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
  SELECT expression_or_column1, expression_or_column2, expression_or_column3
  FROM table
  WHERE clause_identifying_a_single_row
</xsql:set-page-param>
```

The following example sets the page-level parameter to a value selected from database and then uses it as the value of an `xsql:query` attribute.

Setting a Parameter to a Database-Generated Value

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="style.xsl"?>
<page connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:set-page-param name="max-rows-pref">
     SELECT max_rows
     FROM user_profile
     WHERE userid = {@userid}
  </xsql:set-page-param>
  <xsql:query max-rows="{@max-rows-pref}">
    SELECT title, url
    FROM newsstory
    ORDER BY date_entered DESC
  </xsql:query>
</page>
```

33.2.17 <xsql:set-session-param>

Element `<xsql:set-session-param>` is described.

Purpose

Sets an HTTP session-level parameter to a value. The value of the session-level parameter remains for the lifetime HTTP session of the current browser user. The web server controls the session. The value can be supplied by a combination of static text and other parameter values, or from the result of a SQL `SELECT` statement.

Because this feature is specific to Java servlets, this action is effective only if the XSQL page in which it appears is processed by the XSQL servlet. If this action occurs in an XSQL page processed by the XSQL command-line utility or the `XSQLRequest` programmatic API, it does nothing.

Usage Notes

If you use the SQL statement option, the XSQL processor fetches a single row from the result set and assigns the parameter 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.

To set several session parameter values based on the results of a single SQL statement, do not use the `name` attribute. Instead, use the `names` attribute and supply a space-or-comma-delimited list of one or more session parameter names.

Syntax

The syntax for this action is as follows, where *`paramname`* is the name of a parameter and where *`value`* is a value:

```
<xsql:set-session-param name="paramname" value="value"/>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

Alternatively, you can use this syntax, where *SQL_statement* is a SQL SELECT statement and *paramname* is the name of a parameter:

```
<xsql:set-session-param name="paramname">
  SQL_statement
</xsql:set-session-param>
```

Either the name or the names attribute is required. The value attribute and the contained SQL statement are mutually exclusive.

Attributes

Table 33-14 lists the optional attributes supported by this action. Attributes in bold are required; all others are optional.

Table 33-14 Attributes for <xsql:set-session-param>

Attribute Name	Description
name = "*string*"	Name of the session-level variable whose value you want to set. Either use the name or the names attribute, but not both.
names = "*string string ...*"	Space-or-comma-delimited list of the session parameter names whose values you want to set. Either use the name or the names attribute, but not both.
bind-params = "*string*"	Ordered, space-delimited list of one or more XSQL parameter names. The parameter values are used to bind to the JDBC bind variable in the appropriate sequential position in the SQL statement.
error-param = "*string*"	Name of a page-private parameter that is set to the string 'Error' if a nonfatal error occurs while processing this action. Valid value is any parameter name.
ignore-empty-value = "*boolean*"	Indicates whether the session-level parameter assignment is ignored if the value to which it is being assigned is an empty string. Valid values are yes and no (default).
only-if-unset = "*boolean*"	Indicates whether the session variable assignment occurs only when the session variable currently does not exists.Valid values are yes and no (default).
quote-array-values = "*boolean*"	If the parameter name is a simple-valued parameter name (for example, myparam) and if treat-list-as-array="yes" is specified, then specifying quote-array-values="yes" surrounds each string token with single quotation marks before separating the values with commas. Valid values are yes and no (default).
treat-list-as-array = "*boolean*"	Indicates whether the string-value assigned to the parameter is tokenized into an array of separate values before assignment. If any comma is present in the string, then the comma is used for separating tokens. Otherwise, spaces are used.Valid values are yes and no. The default value is yes if the parameter name being set is an array parameter name (for example, myparam[]), and default is no if the parameter name being set is a simple-valued parameter name like myparam.
value = "*string*"	Sets the value to assign to the parameter.

Examples

The following example sets multiple session parameter values based on the results of a single SELECT statement.

Setting Session Parameters

Copyright © 1999, 2025, Oracle and/or its affiliates.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="style.xsl"?>
<page connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:set-session-param names="paramname1 paramname2 paramname3">
    SELECT expression_or_column1, expression_or_column2, expression_or_column3
    FROM   table
    WHERE  clause_identifying_a_single_row
  </xsql:set-session-param>
  <!-- ... -->
</page>
```

33.2.18 <xsql:set-stylesheet-param>

Element `<xsql:set-stylesheet-param>` is described.

Purpose

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 from the result of a SQL `SELECT` statement. The stylesheet parameter is set on any stylesheet used during the processing of the current page.

Usage Notes

If you use the SQL statement option, then 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.

To set several stylesheet parameter values based on the results of a single SQL statement, do not use the `name` attribute. You can use the `names` attribute and supply a space-or-comma-delimited list of one or more stylesheet parameter names.

Syntax

The syntax for this action is as follows, where *`paramname`* is the name of a parameter and where *`value`* is a value:

```
<xsql:set-stylesheet-param name="paramname" value="value"/>
```

Alternatively, you can use this syntax, where *`SQL_statement`* is a SQL `SELECT` statement and *`paramname`* is the name of a parameter:

```
<xsql:set-stylesheet-param name="paramname">
  SQL_statement
</xsql:set-stylesheet-param>
```

Either the `name` or the `names` attribute is required. The `value` attribute and the contained SQL statement are mutually exclusive.

Attributes

Table 33-15 lists the optional attributes supported by this action. Attributes in bold are required; all others are optional.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 33-15 Attributes for <xsql:set-stylesheet-param>

Attribute Name	Description
`name = "string"`	Name of the top-level stylesheet parameter whose value you want to set.
`names = "string string ..."`	Space-or-comma-delimited list of the top-level stylesheet parameter names whose values you want to set. Use the `name` or the `names` attribute, but not both.
`bind-params = "string"`	Ordered, space-delimited list of one or more XSQL parameter names. Parameter values are used to bind to the JDBC bind variable in the appropriate sequential position in the SQL statement.
`error-param = "string"`	Name of a page-private parameter that must be set to the string '`Error`' if a nonfatal error occurs while processing this action. Valid value is any parameter name.
`ignore-empty-value = "boolean"`	Indicates whether the stylesheet parameter assignment is to be ignored if the value to which it is being assigned is an empty string. Valid values are `yes` and `no` (default).
`value = "string"`	Sets the value to assign to the parameter.

Examples

The following example associates a stylesheet and uses the `<xsql:set-stylesheet-param>` action element to assign the value of the XSQL page parameter named `p_table` to the XSLT top-level stylesheet parameter named `table`.

Setting a Stylesheet Parameter

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="style.xsl"?>
<page connname="xmlbook" connection="{@p_connname}">
  <xsql:query null-indicator="yes" xmlns:xsql="urn:oracle-xsql">
  <![CDATA[
    SELECT *
    FROM {@p_table}
    WHERE rownum < 2
  ]>
  </xsql:query>
  <xsql:set-stylesheet-param name="table" value="{@p_table}"
                             xmlns:xsql="urn:oracle-xsql" />
</page>
```

33.2.19 <xsql:update-request>

Element `<xsql:update-request>` is described.

Purpose

Accepts data posted from an XML document or HTML form and uses the XML SQL Utility (XSU) to update the content of an XML document in canonical form from a target table or view.

By combining XSU with XSLT, you can transform XML into the canonical format expected by a given table. Afterward, you can use XSU to update the resulting canonical XML. For a specified database table, the canonical XML form is given by one row of XML output from a `SELECT * ` query against the table.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Syntax

The syntax for this action is:

```
<xsql:update-request table="table_name"/>
```

Attributes

Table 33-3 lists the attributes that you can use on the `<xsql:update-request>` action. Required attributes are in bold.

Table 33-16 Attributes for <xsql:update-request>

Attribute Name	Description
`table = "string"`	Name of the table, view, or synonym to use for updating the XML data.
`key_columns = "string string ..."`	Space-delimited or comma-delimited list of one or more column names. The processor uses the values of these names in the posted XML document 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 updated into canonical ROWSET/ROW format.
`columns = "string"`	Relative or absolute URL of the XSLT transformation to use to transform the document to be updated into canonical ROWSET/ROW format.
`commit = "boolean"`	If set to `yes` (default), invokes `COMMIT` on the current connection after a successful execution of the update. Valid values are `yes` and `no`.
`commit-batch-size = "integer"`	If a positive, nonzero *`integer`* is specified, then after each batch of *`integer`* updated records, the processor issues a `COMMIT`. The default batch size is zero (`0`) if not specified, which means that the processor does not commit interim batches.
`date-format = "string"`	Date format mask to use for interpreting date field values in XML being updated. Valid values are those for the `java.text.SimpleDateFormat` class.
`error-param = "string"`	Name of a page-private parameter that must be set to `Error` if a nonfatal error occurs while processing this action. Valid value is any parameter name.

Examples

The following example parses and transforms the contents of the posted XML document or HTML Form for update.

Updating XML Received in a Parameter

```
<?xml version="1.0"?>
<xsql:update-request table="purchase_order"   key-columns="department_id"
                     connection="demo"        transform="doc-to-departments.xsl"
 xmlns:xsql="urn:oracle-xsql/>
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

34

Oracle XML Developer's Kit Standards

A description is given of the Oracle XML Developer's Kit (XDK) standards.

34.1 XML Standards Supported by XDK

Topics here include XML and Java standards supported by XDK.

34.1.1 Summary of XML Standards Supported by XDK

The XML standards supported by XDK components are described.

Table 34-1 Summary of XML Standards Supported by Oracle XML Developer's Kit

Standard	Java	C	C++
Document Object Model (DOM) Level 1 Specification	Full	Full	Full
Document Object Model Core (2.0)	Full	Full	Full
Document Object Model (DOM) Level 2 Events Specification	Full	Full	Full
Document Object Model (DOM) Level 2 Traversal and Range Specification	Full	Full	Full
Document Object Model (DOM) Level 3 Core Specification	Full	N/A	N/A
Document Object Model (DOM) Level 3 Load and Save Specification	Partial[1]	None	None
Document Object Model (DOM) Level 3 Validation Specification	Full[2]	None	None
JAXP 1.1 and 1.2 (JSR Standard)	Full	N/A	N/A
SAX Project, 1.0, 2.0 core, and 2.0 extension	Full	Full	Full
Extensible Markup Language (XML) 1.0 (Fifth Edition)	Full	Full	Full
XML Base (Second Edition)	Only in XSLT	None	None
Namespaces in XML 1.0 (Third Edition)	Full	Full	Full
XML Pipeline Definition Language Version 1.0	Partial[3]	None	None
XML Schema Part 0: Primer Second Edition	Full	Full[4]	Full[4]
XML Path Language (XPath) Version 1.0	Full	Full	Full
XML Path Language (XPath) 2.0 (Second Edition)	Full	None	None
XML Path Language (XPath) 3.0	Full	None	None
XQuery 1.0: An XML Query Language (Second Edition)	Full	None	None
XQuery and XPath Data Model 3.1	Full	None	None
XPath and XQuery Functions and Operators 3.1	Full	None	None
XQuery 3.0: An XML Query Language	Full	None	None
XQuery and XPath Data Model 3.0	Full	None	None
XPath and XQuery Functions and Operators 3.0	Full	None	None

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table 34-1 (Cont.) Summary of XML Standards Supported by Oracle XML Developer's Kit

Standard	Java	C	C++
XQuery Update Facility 1.0	Full	None	None
XQueryX 3.0	Full	None	None
JSR-000225 XQuery API for Java	Full	None	None
XSL Transformations (XSLT) Version 1.0	Full	Full	Full
XSL Transformations (XSLT) Version 2.0, *Basic XSLT Processor*	Conformance as a basic XSLT processor[5]	None	None

1 DOM Level 3 Load and Save describes the relationship between DOM 3.0 Core and Load and Save.
2 DOM 3.0 Validation describes the relationship between DOM 3.0 Core and Validation.
3 Pipeline Definition Language Standard for XDK for Java describes the parts of the standard that are not supported.
4 The Schema processor fully supports the functionality stated in the specification plus *XML Schema 1.0 Specification Errata*.
5 See XSLT Standard for XDK for Java for details

34.1.2 XML Standards for XDK for Java

Topics here include XDK standards for DOM, XSLT, JAXB, and Pipeline Definition Language.

34.1.2.1 DOM Standard for XDK for Java

The DOM APIs include support for candidate recommendations of DOM Level 3 Validation and DOM Level 3 Load and Save.

> **Note**
>
> In Oracle Database 10g Release 2, XDK for Java implements the candidate recommendation versions of Document Object Model (DOM) Level 3.0 Load and Save and Validation specifications. Oracle plans to produce a release or patch set that will include an implementation of DOM Level 3.0 Load and Save and Validation recommendations. To conform with the recommendations, Oracle might be forced to make changes that are not backward compatible. During this period Oracle does not guarantee backward compatibility with our DOM Load and Save, and Validation implementation. After XDK for Java is updated to conform with the recommendations, standard Oracle policies for backward compatibility will apply to the Oracle DOM Load and Save, and Validation implementation.

34.1.2.1.1 DOM Level 3 Load and Save

The DOM Level 3 Load and Save module enables software developers to load and save Extensible Markup Language (XML) content inside conforming products.

The `charset-overrides-xml-encoding` configuration parameter is not supported by `LSParser`. Optional settings of these configuration parameters are not supported by `LSParser`:

- `disallow-doctype (true)`
- `ignore-unknown-character-denormalizations (false)`

Copyright © 1999, 2025, Oracle and/or its affiliates.

- `namespaces (false)`
- `supported-media-types-only (true)`

The `discard-default-content` configuration parameter is not supported by `LSSerializer`. Optional settings of these configuration parameters are not supported by LSSerializer:

- `canonical-form (true)`
- `format-pretty-print (true)`
- `ignore-unknown-character-denormalizations (false)`
- `normalize-characters (true)`

34.1.2.1.2 DOM 3.0 Validation

DOM 3.0 validation lets users retrieve metadata definitions from XML schemas, query the validity of DOM operations, and validate the DOM documents or subtrees against an XML schema. Because validation is based on a schema, you must convert a document type definition (DTD) to a schema before using these functions.

34.1.2.2 XSLT Standard for XDK for Java

The XDK XSLT processor supports the current recommendations of XSLT 2.0, XPath 2.0, and the shared XPath/XQuery data model.

Oracle XML Development Kit (XDK) supports the XSLT 2.0 (W3C Recommendation, 23 January 2007) as a *basic XSLT processor*, with the following limitation: Support for `xsl:key` and `xsl:sort` behavior is at the XSLT 1.0 level.

See Also

- *Basic XSLT Processor*
- *XSL Transformations (XSLT) Version 1.0*

34.1.2.3 JAXB Standard for XDK for Java

Features not supported by the XDK implementation of the Java Architecture for XML Binding (JAXB) specification are described.

The XDK implementation of the Java Architecture for XML Binding (JAXB) specification does not support these features:

- Javadoc generation
- XML Schema component `any` and substitution groups

34.1.2.4 Pipeline Definition Language Standard for XDK for Java

Differences between the XML Pipeline processor and the W3C Note are presented.

The two differ as follows:

- The parser processes `DOMParserProcess` and `SAXParserProcess` are included in the XML pipeline (Section 1).

Copyright © 1999, 2025, Oracle and/or its affiliates.

- Only the final target output is checked to see if it is up-to-date with the available pipeline inputs. The XML Pipeline processor does not determine whether the intermediate outputs of every process are up-to-date (Section 2.2).
- For the `select` attribute, anything in between double quotation marks ("...") is considered to be a string literal.
- The XML Pipeline processor throws an error if more that one process produces the same infoset (Section 2.4.2.3).
- The `<document>` element is not supported (Section 2.4.2.8).

34.2 Character Sets Supported by XDK

The character sets supported by XDK for Java and XDK for C are described.

34.2.1 Character Sets Supported by XDK for Java

The character-set encodings supported by XDK for Java are described.

XML Schema processor for Java supports documents in these 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

34.2.2 Character Sets Supported by XDK for C

XDK for C supports over 300 Internet Assigned Numbers Authority (IANA) character sets.

These character sets include:

- UTF-8
- UTF-16
- UTF16-BE
- UTF16-LE

Copyright © 1999, 2025, Oracle and/or its affiliates.

- US-ASCII
- ISO-10646-UCS-2
- ISO-8859-{1-9, 13-15}
- EUC-JP
- SHIFT_JIS
- BIG5
- GB2312
- GB_2312-80
- HZ-GB-2312
- KOI8-R
- KSC5601
- EUC-KR
- ISO-2022-CN
- ISO-2022-JP
- ISO-2022-KR
- WINDOWS-{1250-1258}
- EBCDIC-CP-{US,CA,NL,WT,DK,NO,FI,SE,IT,ES,GB,FR,HE,BE,CH,ROECE,YU,IS,AR}
- IBM{037, 273, 277, 278, 280, 284, 285, 297, 420, 424, 437, 500, 775, 850, 852, 855, 857, 858, 860, 861, 863, 865, 866, 869, 870, 871, 1026, 01140, 01141, 01142, 01143, 01144, 01145, 01146, 01147,01148}

You can use any alias of the preceding character sets. In addition, you can use any character set specified in *Oracle AI Database Globalization Support Guide*, except for IW7IS960.

Copyright © 1999, 2025, Oracle and/or its affiliates.

A

XDK for Java XML Error Messages

Error messages are listed for applications that use Oracle XML Developer's Kit (XDK) for Java during the execution of Extensible Markup Language (XML) interfaces.

A.1 XML Parser Error Messages

Extensible Markup Language (XML) parser error messages are in the range XML-20000 through XML-20999.

XML-20003: missing token *string* at line *string*, column *string*

Cause: An expected token was not found in the input data.

Action: Check/update the input data to fix the syntax error.

XML-20004: missing keyword *string* at line *string*, column *string*

Cause: An expected keyword was not found in the input data.

Action: Check/update the input data to the correct keyword.

XML-20005: missing keyword *string* or *string* at line *string*, column *string*

Cause: An expected keyword was not found in the input data.

Action: Check/update the input data to the correct keyword.

XML-20006: unexpected text at line *string*, column *string*; expected EOF

Cause: More text was found after the end-tag of the root element.

Action: The end-tag of the root element can be followed only by comments, PI, or white space. Remove the extra text after the end-tag.

XML-20007: missing content model in element declaration at line *string*, column *string*

Cause: The element declaration was missing the required content model spec. See Production [45] in XML 1.0 2nd Edition.

Action: Add the required content spec to the element declaration.

XML-20008: missing element name in content model at line *string*, column *string*

Cause: The content model in the element declaration was invalid, the content particle requires an element name. See Production [48] in XML 1.0 2nd Edition.

Action: Add the element name to fix the content spec syntactically.

XML-20009: target name *string* of processing instruction at line *string*, column *string* is reserved

Cause: The target names `"XML: xml"`, and so on are reserved for standardization in future versions of XML specification. See Production [17] in XML 1.0 2nd Edition.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Action: If the PI is meant to be XML declaration, make sure the declaration occurs at the very beginning of the file. Otherwise, change to name of the PI.

XML-20010: missing notation name in unparsed entity declaration at line *string*, column *string*

Cause: The notation name used in the unparsed entity declaration did not match the name in a declared notation. See Production [76] in XML 1.0 2nd Edition.

Action: Add the notation declaration to the DTD.

XML-20011: missing attribute type in attribute-list declaration at line *string*, column *string*

Cause: The attribute type was missing the attribute-list declaration. One of these types CDATA, ID, IDREF, IDREFS, ENTITY, ENTITIES, NMTOKEN, or NMTOKENS must be added. See Production [52], [53] in XML 1.0 2nd Edition.

Action: Check and correct attribute declaration.

XML-20012: missing white space at line *string*, column *string*

Cause: The required white space was missing.

Action: Add white space to fix the syntax error.

XML-20013: invalid character *string* in entity value at line *string*, column *string*

Cause: An invalid character was used in the entity value. Characters `&`, `%`, and either `"` or `'` (based on the value delimiters) are invalid. See Production [9] in XML 1.0 2nd Edition.

Action: Use entity or character references instead of the characters For example, `&` or `&` can be used instead of `&`.

XML-20014: -- not allowed in comment at line *string*, column *string*

Cause: A syntax error in comment due to the use of "--"See Production [15] in XML 1.0 2nd Edition.

Action: Fix the comment, and use `--` only as part of end of comment `-->`

XML-20015:]> not allowed in text at line *string*, column *string*

Cause: `]>` is not allowed in text. It is used only as end marker for CDATA Section. See Production [14] in XML 1.0 2nd Edition.

Action: Fix the text content by using `>` or char ref for `>`.

XML-20016: white space not allowed before occurrence indicator at line *string*, column *string*

Cause: White space is not allowed in the contentspec before the occurrence indicator. For example, `<!ELEMENT x (a,b) *>` is not valid. See Production [47], [48] in XML 1.0 2nd Edition.

Action: Fix the contentspec by removing the extra space

XML-20017: occurrence indicator *string* not allowed in mixed-content at line *string*, column *string*

Cause: Occurrence is not allowed in mixed content declaration. For example, `<!ELEMENT x (#PCDATA)?>` is not valid. See Production [51] in XML 1.0 2nd Edition.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Action: Fix the syntax to remove the occurrence indicator.

XML-20018: content list not allowed inside mixed-content at line *string*, column *string*

Cause: Content list is not allowed in mixed-content declaration. For example, `<!ELEMENT x (#PCDATA | (a,b))>` is not valid. See Production [51] in XML 1.0 2nd Edition.

Action: Fix the syntax to remove the content list.

XML-20019: duplicate element *string* in mixed-content declaration at line *string*, column *string*

Cause: Duplicate element name was found in mixed-content declaration. For example, `<! ELEMENT x (#PCDATA | a | a)>` is not valid. See Production [51] in XML 1.0 2nd Edition

Action: Remove the duplicate element name.

XML-20020: root element *string* does not match the DOCTYPE name *string* at line *string*, column *string*

Cause: failed: The name in the document type declaration must match the element type of the root element. For example: `<?xml version="1.0"?> <!DOCTYPE greeting [ <!ELEMENT greeting (#PCDATA)> ]> <salutation>Hello!</salutation>`. The document's root element, `salutation`, does not match the root element declared in the DTD (`greeting`).

Action: Correct the document.

XML-20021: duplicate element declaration *string* at line *string*, column *string*

Cause: Element was declared twice in the DTD.

Action: Remove the duplicate declaration.

XML-20022: element *string* has multiple ID attributes at line *string*, column *string*

Cause: failed: No element type may have more than one ID attribute specified.

Action: Correct the document, by removing the duplicate ID attribute `decl`.

XML-20023: ID attribute *string* in element *string* must be #IMPLIED or #REQUIRED at line *string*, column *string*

Cause: failed: An ID attribute must have a declared default of `#IMPLIED` or `#REQUIRED`.

Action: Fix the attribute declaration.

XML-20024: missing required attribute *string* in element *string* at line *string*, column *string*

Cause: failed: If the default declaration is the keyword `#REQUIRED`, then the attribute must be specified for all elements of the type in the attribute-list declaration.

Action: Fix the input document by specifying the required attribute.

XML-20025: duplicate ID value: *string*

Cause: Values of type ID must match the Name production. A name must not appear more than once in an XML document as a value of this type; thus, ID values must uniquely identify the elements which bear them.

Action: Fix the input document by removing the duplicate ID value.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-20026: undefined ID value *string* in IDREF

Cause: failed "Values of type IDREF must match value of some ID attribute.

Action: Fix the document by adding an ID corresponding the to the IDREF, or removing the IDREF.

XML-20027: attribute *string* in element *string* has invalid enumeration value *string* at line *string*, column *string*

Cause: failed: Values of this type must match one of the Nmtoken tokens in the declaration.

Action: Fix the attribute value to match one of the enumerated values.

XML-20028: attribute *string* in element *string* has invalid value *string*, must be *string* at line *string*, column {5}

Cause: failed: If an attribute has a default value declared with the `#FIXED` keyword, instances of that attribute must match the default value.

Action: Update the attribute value to match the fixed default value.

XML-20029: attribute default must be REQUIRED, IMPLIED, or FIXED at line *string*, column *string*

Cause: The declared default value must meet the lexical constraints o the declared attribute type.

Action: Use one of `REQUIRED`, `IMPLIED`, or `FIXED` for attribute default `decl`.

XML-20030: invalid text in content of element *string* at line *string*, column *string*

Cause: The element does not allow text in content. An element is valid if there is a declaration matching element `decl` where the Name matches the element type, and one of these holds:

The declaration matches children and the sequence of child elements belongs to the language generated by the regular expression in the content model, with optional white space (characters matching the nonterminal S) between the start-tag and the first child element, between child elements, or between the last child element and the end-tag. A CDATA section containing only white space does not match the nonterminal S, and hence cannot appear in these positions.

Action: Fix the content by removing unexpected text.

XML-20031: invalid element *string* in content of element *string* at line *string*, column *string*

Cause: The element has invalid content. An element is valid if there is a declaration matching element `decl` where the Name matches the element type, and one of these holds:

1. The declaration matches children and the sequence of child elements belongs to the language generated by the regular expression in the content model, with optional white space (characters matching the nonterminal S) between the start-tag and the first child element, between child elements, or between the last child element and the end-tag. A CDATA section containing only white space does not match the nonterminal S, and hence cannot appear in these positions.
2. The declaration matches Mixed and the content consists of character data and child elements whose types match names in the content model.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Action: Fix the content by removing unexpected elements.

XML-20032: incomplete content in element *string* at line *string*, column *string*

Cause: The element has invalid content. An element is valid if there is a declaration matching element `decl` where the Name matches the element type, and one of these holds:

1. The declaration matches children and the sequence of child elements belongs to the language generated by the regular expression in the content model, with optional white space (characters matching the nonterminal S) between the start-tag and the first child element, between child elements, or between the last child element and the end-tag. A CDATA section containing only white space does not match the nonterminal S, and hence cannot appear in these positions.
2. The declaration matches Mixed and the content consists of character data and child elements whose types match names in the content model.

Action: Fix the content by removing unexpected elements.

XML-20033: invalid replacement-text for entity *string* at line *string*, column *string*

Cause: Parameter-entity replacement text must be properly nested with markup declarations. That is to say, if either the first character or the last character of a markup declaration (markup `decl` above) is contained in the replacement text for a parameter-entity reference, both must be contained in the same replacement text.

Action: Fix the entity value.

XML-20034: end-element tag *string* does not match start-element tag *string* at line *string*, column *string*

Cause: The Name in an element's end-tag must match the element type in the start-tag.

Action: Fix the end-tag or start-tag to match the other.

XML-20035: duplicate attribute *string* in element *string* at line *string*, column *string*

Cause: No attribute name may appear more than once in the same start-tag or empty-element tag.

Action: Remove the duplicate attribute.

XML-20036: invalid character *string* in attribute value at line *string*, column *string*

Cause: An invalid character was used in the attribute value, the characters `&`, `<`, and either `"` or `'` (based on the value delimiters) are invalid. See Production [10] in XML 1.0 2nd Edition.

Action: Use entity or character references instead of the characters For example, `&` or `&` can be used instead of `&`.

XML-20037: invalid reference to external entity *string* in attribute *string* at line *string*, column *string*

Cause: Attribute values cannot contain direct or indirect entity references to external entities.

Action: Fix document to remove reference to external entity in attribute.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-20038: invalid reference to unparsed entity *string* in element *string* at line *string*, column *string*

Cause: An entity reference must not contain the name of an unparsed entity. Unparsed entities may be referenced only in attribute values declared to be of type ENTITY or ENTITIES.

Action: Fix document to remove reference to unparsed entity in content.

XML-20039: invalid attribute type *string* in attribute-list declaration at line *string*, column *string*

Cause: Invalid attribute type was used in the attribute-list declaration. One of these types CDATA, ID, IDREF, IDREFS, ENTITY, ENTITIES, NMTOKEN, or NMTOKENS must be added. See Production [52], [53] in XML 1.0 2nd Edition.

Action: Check and correct attribute declaration.

XML-20040: invalid character *string* in element content at line *string*, column *string*

Cause: Characters referred to using character references must match the production for Char.

Action: Fix the document by removing the invalid character or char-ref.

XML-20041: entity reference *string* refers to itself at line *string*, column *string*

Cause: A parsed entity must not contain a recursive reference to itself, either directly or indirectly.

Action: Fix the document.

XML-20042: invalid Nmtoken: *string*

Cause: Values of this type must match one of the Nmtoken tokens in the declaration, and must be valid Nmtoken"

Action: Fix the attribute value.

XML-20043: invalid character *string* in public identifier at line *string*, column *string*

Cause: Invalid character used in public identifier. See Production [12], [13] in XML 1.0 2nd Edition.

Action: Fix the public identifier.

XML-20044: undeclared namespace prefix *string* used at line *string*, column *string*

Cause: The prefix was not defined in any namespace declaration in scope.

Action: Add a namespace declaration to define the prefix.

XML-20045: attribute *string* in element *string* must be an unparsed entity at line *string*, column *string*

Cause: Values of type ENTITY must match the Name production, values of type ENTITIES must match Names; each Name must match the name of an unparsed entity declared in the DTD.

Action: Fix the attribute value to refer to an unparsed entity.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-20046: undeclared notation *string* used in unparsed entity *string* at line *string*, column *string*

Cause: Values of this type must match one of the notation names included in the declaration; all notation names in the declaration must be declared.

Action: Fix the notation name in the unparsed entity declaration.

XML-20047: missing element declaration *string*

Cause: The element declaration referred to by an attribute declaration was not found in the DTD.

Action: Fix the DTD by adding the element declaration.

XML-20048: duplicate entity declaration *string* at line *string*, column *string*

Cause: Warning regarding duplicate entity declaration.

Action: No action required.

XML-20049: invalid use of NDATA in parameter entity declaration at line *string*, column *string*

Cause: `NDATA` declaration was found in parameter entity declaration. It is allowed only in general unparsed entity declaration. See Production [72], [74] in XML 1.0 2nd Edition.

Action: Fix the entity declaration.

XML-20050: duplicate attribute declaration *string* at line *string*, column *string*

Cause: Warning regarding duplicate attribute declaration.

Action: No action required.

XML-20051: duplicate notation declaration *string* at line *string*, column *string*

Cause: Only one notation declaration can declare a given Name.

Action: Fix the document by removing the duplicate notation.

XML-20052: undeclared attribute *string* used at line *string*, column *string*

Cause: The attribute declaration was not found in the DTD.

Action: Fix the DTD by adding the attribute declaration.

XML-20053: undeclared element *string* used at line *string*, column *string*

Cause: The element declaration was not found in the DTD.

Action: Fix the DTD by adding the element declaration.

XML-20054: undeclared entity *string* used at line *string*, column *string*

Cause: The entity declaration was not found in the DTD.

Action: Fix the DTD by adding the element declaration.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-20055: invalid document returned by NodeFactory's createDocument

Cause: The document returned by `createDocument` function of `NodeFactory` was invalid, either it was null or instance of an unsupported class.

Action: Fix `NodeFactory` implementation to return an instance of `XMLDocument` or its subclass.

XML-20056: invalid SAX feature *string*

Cause: The SAX feature supplied was not a valid feature name.

Action: See the documentation for a valid list of features.

XML-20057: invalid value *string* passed for SAX feature *string*

Cause: The value supplied for the SAX feature was not valid.

Action: See the documentation for a valid list of features and their corresponding values.

XML-20058: invalid SAX property *string*

Cause: The SAX property supplied was not a valid property name.

Action: See the documentation for a valid list of properties.

XML-20059: invalid value passed for SAX property *string*

Cause: The value supplied for the SAX property was not valid.

Action: See the documentation for a valid list of properties and their corresponding values

XML-20060: Error occurred while opening URL *string*

Cause: An error occurred while opening the supplied URL.

Action: Verify the URL, and take appropriate action to allow data to be read.

XML-20061: invalid byte stream *string* in UTF8 encoded data

Cause: The input data contained bytes that are not valid with respect to UTF-8 encoding scheme.

Action: Fix the input data.

XML-20062: 5-byte UTF8 encoding not supported

Cause: The XML Parser does not support 5-byte UTF-8 encoding scheme. It is also possible that invalid UTF-8 characters were misinterpreted as 5-byte UTF-8 encoding.

Action: If the data contains invalid UTF-8 bytes, fix the input, otherwise if 5-byte UTF-8 supported is required, contact Oracle Support.

XML-20063: 6-byte UTF8 encoding not supported

Cause: The XML Parser does not support 6-byte UTF-8 encoding scheme. It is also possible that invalid UTF-8 characters were misinterpreted as 6-byte UTF-8 encoding.

Action: If the data contains invalid UTF-8 bytes, fix the input, otherwise if 6-byte UTF-8 supported is required, contact Oracle Support.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-20064: invalid XML character *string*

Cause: Invalid XML character was found in the input data.

Action: Fix the input data.

XML-20065: encoding *string* doesn't match encoding *string* in XML declaration

Cause: The encoding of the data (either by auto-detection or user supplied)didn't match the encoding specified in the XML declaration.

Action: Fix the XML declaration to match the encoding of the data.

XML-20066: encoding *string* not supported

Cause: The XML Parser does not support the specified encoding.

Action: If the support for the encoding is required, contact Oracle Support.

XML-20067: invalid InputSource returned by EntityResolver's resolveEntity

Cause: An invalid instance of `InputSource` was returned by the `EntityResolverAn InputSource` can be invalid if the none of `Reader`, `InputStream`, and `SystemId` were initialized or if the `SystemId` was invalid.

Action: Fix the `EntityResolver` class to return a valid instance of `InputSource`.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-20100: Expected *string*.

XML-20101: Expected *string* or *string*.

XML-20102: Expected *string*, *string*, or *string*.

XML-20103: Illegal token in content model.

XML-20104: Could not find element with ID *string*.

XML-20105: ENTITY type Attribute value *string* does not match any unparsed Entity.

XML-20106: Could not find Notation *string*.

XML-20107: Could not find declaration for element *string*.

XML-20108: Start of root element expected.

XML-20109: PI with the name 'xml' can occur only in the beginning of the document.

XML-20110: #PCDATA expected in mixed-content declaration.

XML-20111: Element *string* repeated in mixed-content declaration.

XML-20112: Error opening external DTD *string*.

XML-20113: Unable to open input source (*string*).

XML-20114: Bad conditional section start syntax, expected '['.

XML-20115: Expected ']>' to end conditional section.

XML-20116: Entity *string* already defined, using the first definition.

XML-20117: NDATA not allowed in parameter entity declaration.

XML-20118: NDATA value required.

XML-20119: Entity Value should start with quote.

XML-20120: Entity value not well-formed.

XML-20121: End tag does not match start tag *string*.

XML-20122: '=' missing in attribute.

XML-20123: '>' Missing from end tag.

XML-20124: An attribute cannot appear more than once in the same start tag.

XML-20125: Attribute value should start with quote.

XML-20126: '<' cannot appear in attribute value.

XML-20127: Reference to an external entity not allowed in attribute value.

XML-20128: Reference to unparsed entity not allowed in element content.

XML-20129: Namespace prefix *string* used but not declared.

XML-20130: Root element name must match the DOCTYPE name.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-21999.

XML-21000: invalid size *string* specified

Cause: An invalid size or count was passed to a DOM function.

Action: Correct the argument passed to a valid value.

XML-21001: invalid index *string* specified; must be between 0 and *string*

Cause: An invalid index was passed to a DOM function.

Action: Correct the argument passed to a valid value specified by the bounds in the error messag

XML-21002: cannot add an ancestor as a child node

Cause: The DOM operation was trying to a add an ancestor node as a child. This can lead to inconsistencies in the tree, so it is not allowed.

Action: Check the application to fix the usage.

XML-21003: node of type *string* cannot be added to node of type *string*

Cause: The DOM specification does not allow the parent-child combinationused in the DOM operation.

Action: See the DOM specification to fix the usage.

XML-21004: document node can have only one *string* node as child

Cause: The XML well-formedness requires that the document node have onlyone element node as its child. The application tried adding addinga second element node.

Action: Fix usage in the application.

XML-21005: node of type *string* cannot be added to attribute list

Cause: The attribute list (instance of NamedNodeMap) can contain onlyattribute nodes.

Action: Fix usage of NamedNodeMap.

XML-21006: cannot add a node belonging to a different document

Cause: The node being added was created by a different document. The DOMspecification does not allow use of nodes across documents.

Action: Use importNode or adoptNode to move a node from one document to another, before adding it.

XML-21007: invalid character *string* in name

Cause: The qualified or local name passed was invalid.

Action: Fix the name to contain only valid

XML-21008: cannot set value for node of type *string*

Cause: The node of the specified type cannot have value.

Action: Fix usage of DOM functions.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-21009: cannot modify descendants of entity or entity reference nodes

Cause: The descendants of entity or entity reference nodes are read-onlynodes, and modification is not allowed.

Action: Fix usage of DOM functions.

XML-21010: cannot modify DTD's content

Cause: DTD and all its content is read-only and cannot be modified.

Action: Fix usage of DOM functions.

XML-21011: cannot remove attribute; not found in the current element

Cause: An attempt was made to remove an attribute that does not belong thecurrent element.

Action: Fix usage in application.

XML-21012: cannot remove or replace node; it is not a child of the current node

Cause: An attempt was made to remove an node that does not belong thecurrent node as a child.

Action: Fix usage in application.

XML-21013: parameter *string* not recognized

Cause: The DOM parameter was not recognized.

Action: See the documentation for a valid list of parameters.

XML-21014: value *string* of parameter *string* is not supported

Cause: The DOM parameter was not recognized.

Action: See the documentation for a valid list of parameters.

XML-21015: cannot add attribute belonging to another element

Cause: An attempt was made to add an attribute that belonged theanother element.

Action: Fix usage in application.

XML-21016: invalid namespace *string* for prefix *string*

Cause: The namespace for xml, and xmlns prefixes is fixed, and usage mustmatch these.

Action: Correct the namespace for the prefixes, namespaces are xml = http://www.w3.org/XML/1998/namespace xmlns = http://www.w3.org/2000/xmlns/

XML-21017: invalid qualified name: *string*

Cause: The qualified name passed to a DOM function was invalid.

Action: Fix the qualified name.

XML-21018: conflicting namespace declarations *string* and *string* for prefix *string*

Cause: The DOM tree has conflicting namespace declarations for the sameprefix. Such a DOM tree cannot be serialized.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Action: Fix the DOM tree, before printing it.

XML-21019: *string* object is detached

Cause: The object was detached, no operations are supported ona detached object. The object can be a Range or iterator object

Action: Fix the usage in application.

XML-21020: bad boundary specified; cannot partially select a node of type *string*

Cause: The boundary specified in the range was invalid. The selectioncan be partial only for text nodes.

Action: Fix the usage in the application.

XML-21021: node of type *string* does not support range operation *string*

Cause: The range operation is not supported on the node type specified.

Action: See the DOM documentation for restrictions of node types for each range operation.

XML-21022: invalid event type: *string*

Cause: The event type passed was invalid.

Action: Fix usage in the application.

XML-21023: prefix not allowed on nodes of type *string*

Cause: The application tried to set prefix on a node on which prefix is notallowed

Action: Fix usage in the application.

XML-21024: import not allowed on nodes of type *string*

Cause: The application tried to import a node of type DOCUMENT orDOCUMENT FRAGMENT.

Action: Fix usage in the application.

XML-21025: rename not allowed on nodes of type *string*

Cause: The application tried to import a node of type other than ELEMENT orATTRIBUTE.

Action: Fix usage in the application.

XML-21026: Unrepresentable character in node: *string*

Cause: A node contains an invalid character, eg. CDATA section contain a termination character.

Action: Set appropriate DOMConfiguration parameter.

XML-21027: Namespace normalization error in node: *string*

Cause: Namespace fixup cannot be performed on this node.

Action: Set namespace normalization to false.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-21997: function not supported on THICK DOM

Cause: A function on THICK (for example, XDB based) DOM which is not supported was called.

Action: See the XDK documentation for possible alternatives for functions not supported on THICK DOM.

XML-21998: system error occurred: *string*

Cause: Non-DOM related system errors occurred.

Action: Check with ORA error(s) embedded in the message and consult with developers for possible causes.

Copyright © 1999, 2025, Oracle and/or its affiliates.

A.3 XSLT Error Messages

Extensible Stylesheet Language Transformation (XSLT) error messages are in the range XML-22000 through XML-22999.

XML-22000: Error while parsing XSL file (*string*).

XML-22001: XSL Stylesheet does not belong to XSLT namespace.

XML-22002: Error while processing include XSL file (*string*).

XML-22003: Unable to write to output stream (*string*).

XML-22004: Error while parsing input XML document (*string*).

XML-22005: Error while reading input XML stream (*string*).

XML-22006: Error while reading input XML URL (*string*).

XML-22007: Error while reading input XML reader (*string*).

XML-22008: Namespace prefix *string* used but not declared.

XML-22009: Attribute *string* not found in *string*.

XML-22010: Element *string* not found in *string*.

XML-22011: Cannot construct XML PI with content: *string*.

XML-22012: Cannot construct XML comment with content: *string*.

XML-22013: Error in expression: *string*.

XML-22014: Expecting node-set before relative location path.

XML-22015: Function *string* not found.

XML-22016: Extension function namespace should start with *string*.

XML-22017: Literal expected in *string* function. Found *string*.

XML-22018: Parse Error in *string* function.

XML-22019: Expected *string* instead of *string*.

XML-22020: Error in extension function arguments.

XML-22021: Error parsing external document: *string*.

XML-22022: Error while testing predicates. Not a nodeset type.

XML-22023: Literal Mismatch.

XML-22024: Unknown multiply operator.

XML-22025: Expression error: Empty string.

XML-22026: Unknown expression at EOF: *string*.

XML-22027: Closing } not found in Attribute Value template.

Copyright © 1999, 2025, Oracle and/or its affiliates.

***string* element.**

XML-22049: Template *string* invoked but not defined.

XML-22050: Duplicate variable *string* definition.

XML-22051: only a literal or a reference to a variable or parameter is allowed in id() function when used as a pattern

XML-22052: no sort key named as: *string* was defined

XML-22053: cannot detect encoding in unparsed-text(), please specify

XML-22054: no such xsl:function with namespace: *string* and local name: *string* was defined

XML-22055: range expression can only accept xs:integer data type, but not *string*

XML-22056: exactly one of four group attributes must be present in xsl:for-each-group

XML-22057: *string* can only have *string* as children

XML-22058: wrong child of xsl:function

XML-22059: wrong child order of xsl:function

XML-22060: TERMINATE PROCESSING

XML-22061: teminate attribute in <xsl:message> can only be yes or no

XML-22062: *string* must have at least one *string* child

XML-22063: no definition for character-map with qname *string*

XML-22064: cannot define character-map with the same name *string* and the same import precedence

Cause: A required child was not found.

Action: After error `mesgfreeze` is over, throws an error (without the required child element, it can do nothing).

XML-22065: at least one *string* must be defined under *string*

Cause: A required child is missing.

Action: Without the required child, it can do nothing.

XML-22066: if select attribute is present, *string* instructions sequence-constructor must be empty

Cause: Attribute and sequence constructor `select` must be mutually exclusive for this instruction.

Action: None.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-22067: if use attribute is present, *string* instructions sequence-constructor must be empty

Cause: Attribute and sequence constructor use must be mutually exclusive for this instruction.

Action: None.

XML-22068: only primary sort key is allowed to have the stable attribute.

Cause: The secondary sort key has a stable attribute.

Action: None.

XML-22069: only *string* or *string* is allowed.

Cause: User typo.

Action: None.

XML-22101: DOMSource node as this type not supported.

XML-22103: DOMResult can not be this kind of node.

XML-22106: Invalid StreamSource - InputStream, Reader, and SystemId are null.

XML-22107: Invalid SAXSource - InputSource is null.

XML-22108: Invalid Source - URL format is incorrect.

XML-22109: Internal error while reporting SAX events.

XML-22110: Invalid StreamResult set in TransformerHandler.

XML-22111: Invalid Result set in TransformerHandler.

XML-22112: Namespace URI missing }.

XML-22113: Namespace URI should start with {.

XML-22117: URL format has problems (null or bad format or missing 'href' or missing '=').

XML-22121: Could not get associated stylesheet.

XML-22122: Invalid StreamResult - OutputStream, Writer, and SystemId are null.

XML-22900: An internal error condition occurred.

A.4 XPath Error Messages

XPath error messages are in the range XML-23000 through XML-23999.

XML-23002: internal xpath error

Cause: This was an error returned by the XPath/XQuery datamodel or XPath F&O.

Action: Check the XPath expression.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-23003: XPath 2.0 feature schema-element/schema-attribute not supported

Cause: This error was caused by using the kindtest schema-element or schema-attribute. These are not supported for this release.

Action: Remove usage of schema-element or schema-attribute kindtest

XML-23006: value does not match required type

Cause: During the evaluation phase, there was a type error as the value did not match a required type specified by the matching rules in XPath 2.0 `SequenceType` matching.

Action: Modify the stylesheet to reflect the correct type.

XML-23007: FOAR0001: division by zero

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23008: FOAR0002: numeric operation overflow/unflow

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23009: FOCA0001: Error in casting to decimal

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23010: FOCA0002: invalid lexical value

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23011: FOCA0003: input value too large for integer

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23012: FOCA0004: Error in casting to integer

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23013: FOCA0005: NaN supplied as float/double value

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23014: FOCH0001: invalid codepoint

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-23015: FOCH0002: unsupported collation

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23016: FOCH0003: unsupported normalization form

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23017: FOCH0004: collation does not support collation units

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23018: FODC0001: no context document

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23019: FODC0002: Error retrieving resource

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23020: FODC0003: Error parsing contents of resource

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23021: FODC0004: invalid argument to fn:collection()

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23022: FODT0001: overflow in date/time arithmetic

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23023: FODT0002: overflow in duration arithmetic

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23024: FONC0001: undefined context item

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-23025: FONS0002: default namespace is defined

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23026: FONS0003: no prefix defined for namespace

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23027: FONS0004: no namespace found for prefix

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23028: FONS0005: base URI not defined in the static context

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23029: FORG0001: invalid value for cast/constructor

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23030: FORG0002: invalid argument to fn:resolve-uri()

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23031: FORG0003: zero-or-one called with sequence containing more than one item

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23032: FORG0004: fn:one-or-more called with sequence containing no items

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23033: FORG0005: exactly-one called with sequence containing zero or more than one item

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23034: FORG0006: invalid argument type

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-23035: FORG0007: invalid argument to aggregate function

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23036: FORG0008: both arguments to fn:dateTime have a specified timezone

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23037: FORG0009: base uri argument to fn:resolve-uri is not an absolute URI

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23038: FORX0001: invalid regular expression flags

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23039: FORX0002: invalid regular expression

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23040: FORX0003: regular expression matches zero-length string

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23041: FORX0004: invalid replacement string

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23042: FOTY0001: type error

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23043: FOTY0011: context item is not a node

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23044: FOTY0012: items not comparable

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-23045: FOTY0013: type does not have equality defined

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23046: FOTY0014: type exception

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23047: FORT0001: invalid number of parameters

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23048: FOTY0002: type definition not found

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23049: FOTY0021: invalid node type

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23050: FOER0000: unidentified error

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23051: FODC0005: invalid argument to fn:doc

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

XML-23052: FODT0003: invalid timezone value

Cause: This was an XPath 2.0 F&O specification error.

Action: Check the XPath expression.

A.5 XML Schema Validation Error Messages

XML schema validation error messages are in the range XML-24000 through XML-24099.

XML-24000: internal error

Cause: An unexpected error occurred during processing.

Action: Report the error.

XML-24001: attribute *string* not expected at line *string*, column *string*

Cause: [cvc-assess-attr.1] The attribute were not expected for owner element.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Action: Add the attribute declaration to the type of the owner element.

XML-24002: can not find element declaration *string*.

Cause: [cvc-assess-elt.1.1.1.1]The element declaration required by processorfor validation was absent.

Action: Add the element declaration to schema, or change the instance document to comply to schema.

XML-24003: context-determined element declaration *string* absent.

Cause: [cvc-assess-elt.1.1.1.2] The element declaration required by context was missing in schema.

Action: Add the element declaration to schema.

XML-24004: declaration for element *string* absent.

Cause: [cvc-assess-elt.1.1.1.3] The context-determined declaration was not skip and the declaration that matches the element could not be found in schema.

Action: Add the element declaration to schema or change the context-determined declaration to skip.

XML-24005: element *string* not assessed

Cause: [cvc-assess-elt.2]

XML-24006: element *string* laxly assessed

Cause: [cvc-assess-elt.2]

XML-24007: missing attribute declaration *string*in element *string*

Cause: [cvc-attribute.1] Attribute declaration was absent from element declaration.

Action: Add the attribute declaration to schema.

XML-24008: type absent for attribute *string*

Cause: [cvc-attribute.2] Missing type definition for the attribute declaration.

Action: Specify a data type for the attribute declaration.

XML-24009: invalid attribute value *string*

Cause: [cvc-attribute.3] Invalid attribute value with respect to its type.

Action: Correct the attribute value in instance.

XML-24010: attribute value *string* and fixed value *string* not match

Cause: [cvc-au] Attribute's normalized value was not the same as the fixed value declared.

Action: Change attribute value to the required value.

XML-24011: type of element *string* is abstract.

Cause: [cvc-complex-type.1] The type of this element was specified as abstract.

Action: Remove the abstract attribute from the type definition.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-24012: no children allowed for element *string* with empty content type

Cause: [cvc-complex-type.2.1] The content type was specified empty while the actual content was not.

Action: Make the content empty or modify the content type of this element.

XML-24013: element child *string* not allowed for simple content

Cause: [cvc-complex-type.2.2] Element was declared with simple content, but instance had element children.

Action: Use only character content for this element.

XML-24014: characters *string* not allowed for element-only content

Cause: [cvc-complex-type.2.3] Characters appeared in the content of element with element-only content.

Action: Use only element children for this element.

XML-24015: multiple ID attributes in element *string* at line *string*, column *string*

Cause: [cvc-complex-type.2.5] More than one attributes with type ID or its derivation matched attribute wildcard.

Action: Do not use more than one attributes with ID or ID derived type.

XML-24016: invalid string value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not valid with respect to string type.

Action: Correct the value to satisfy the declared type.

XML-24017: invalid boolean value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not valid with respect to boolean type.

Action: Correct the value to satisfy Boolean type, valid values are `"0: 1"`, `"true"`, and `"false"`.

XML-24018: invalid decimal value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters could not be parsed into a decimal value.

Action: Correct the data value to satisfy decimal type.

XML-24019: invalid float value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters could not be parsed into a float value.

Action: Correct the value to satisfy string type

XML-24020: invalid double value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not in valid double format as specified in IEEE 754-1985.

Action: Correct the value to satisfy double format.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-24021: invalid duration *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not in correct extended date time format defined in ISO 8601.

Action: Correct the value to satisfy format `PnYnMnDTnHnMnS`.

XML-24022: invalid date value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not in valid calendar date format specified in ISO 8601.

Action: Correct the value to satisfy `CCYY-MM-DD` format.

XML-24023: invalid dateTime value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not in valid combined data time format as specified in ISO 8601.

Action: Correct the value to satisfy format `CCYY-MM-DDThh:mm:ss` with optional time zone.

XML-24024: invalid time value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not in valid time format as specified in ISO 8601.

Action: Correct the value to satisfy format `DDThh:mm:ss` with optional time zone.

XML-24025: invalid gYearMonth value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not in valid right-truncated date format, as specified in ISO 8601.

Action: Correct the value to satisfy format `CCYY-MM`.

XML-24026: invalid gYear value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not in valid right-truncated date format, as specified in ISO 8601.

Action: Correct the value to satisfy format `CCYY`.

XML-24027: invalid gMonthDay value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not in valid left-truncated date format, as specified in ISO 8601.

Action: Correct the value to required format `--MM-DD`.

XML-24028: invalid gDay value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not in valid left-truncated date format, as specified in ISO 8601.

Action: Correct the value to required format `---DD`.

XML-24029: invalid gMonth value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not in valid left-and-right-truncated date format, as specified in ISO 8601.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Action: Correct the value to required format `--MM--`.

XML-24030: invalid hexBinary value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not valid hex encoded binary.

Action: Correct the value to satisfy `hexBinary` type.

XML-24031: invalid base64Binary value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not valid with respect to base64 encoding.

Action: Correct the value to satisfy base64 binary encoding.

XML-24032: invalid anyURI value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not in valid format as specified in RFC 2396 and RFC 2732.

Action: Correct the value to satisfy anyURI type.

XML-24033: invalid QName value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not in valid QName format.

Action: Correct the value to satisfy QName type.

XML-24034: invalid NOTATION value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not valid value for `NOTATION` type.

Action: Correct the value to satisfy `NOTATION` type.

XML-24035: invalid normalizedString value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not valid `normalizedString` value.

Action: Correct the value to satisfy `normalizedString` type.

XML-24036: invalid token value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not valid value for token type.

Action: Correct the value to satisfy token type.

XML-24037: invalid language value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not valid value for language type.

Action: Correct the value to satisfy language type.

XML-24038: invalid NMTOKEN value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not valid value for `NMTOKEN` type.

Action: Correct the value to satisfy `NMTOKEN` type.

XML-24039: invalid NMTOKENS value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not valid list of `NMTOKEN` type.

Action: Correct the value to satisfy `NMTOKENS` type.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-24040: invalid Name value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not valid value for `Name` type.

Action: Correct the value to satisfy `Name` type.

XML-24041: invalid NCName value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not valid value for `NCName` type.

Action: Correct the value to satisfy `NCName` type.

XML-24042: invalid ID value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not valid value for `ID` type.

Action: Correct the value to satisfy `ID` type.

XML-24043: invalid IDREF value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not valid value for `IDREF` type.

Action: Correct the value to satisfy `IDREF` type.

XML-24044: invalid ENTITY value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not valid value for `ENTITY` type

Action: Correct the value to satisfy `ENTITY` type.

XML-24045: invalid ENTITIES value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not valid list of `ENTITY` value.

Action: Correct the value to satisfy `ENTITIES` type.

XML-24046: invalid integer value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not valid value for integer type.

Action: Correct the value to satisfy integer type.

XML-24047: invalid nonPositiveInteger value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not valid value for `nonPositiveInteger` type.

Action: Correct the value to satisfy `nonPositiveInteger` type.

XML-24048: invalid negativeInteger value *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not valid value for `negativeInteger` type.

Action: Correct the value to satisfy `negativeInteger` type.

XML-24049: invalid long value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not valid value for `long` type.

Action: Correct the value to satisfy `long` type.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-24050: invalid int value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not valid value for `int` type.

Action: Correct the value to satisfy `int` type.

XML-24051: invalid short value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not valid value for `short` type.

Action: Correct the value to satisfy `short` type.

XML-24052: invalid byte value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not valid value for `byte` type.

Action: Correct the value to satisfy `byte` type.

XML-24053: invalid nonNegativeInteger value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not valid value for `nonNegativeInteger` type.

Action: Correct the value to satisfy `nonNegativeInteger` type.

XML-24054: invalid unsignedLong value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not valid value for `unsignedlong` type.

Action: Correct the value to satisfy `unsignedlong` type.

XML-24055: invalid unsignedInt value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not valid value of `unsignedInt` type.

Action: Correct the value to satisfy `unsignedInt` type.

XML-24056: invalid unsignedShort value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not valid value for `unsignedShort` type.

Action: Correct the value to satisfy `unsignedShort` type.

XML-24057: invalid unsignedByte value *string* at line *string*, column *string*

Cause: [cvc-datatype-valid.1.2.2] Characters were not valid value for `unsignedByte` type.

Action: Correct the value to satisfy `unsignedByte` type.

XML-24058: value *string* must be valid with respect to one member type

Cause: [cvc-datatype-valid.1.2.3] Characters were invalid with respect to any member type of union.

Action: Correct data value to satisfy at least one member type.

XML-24059: element *string* not expected at line *string*, column *string*

Cause: [cvc-elt.1]

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-24060: element *string* abstract

Cause: [cvc-elt.2] Element declared abstract was used in instance document.

Action: Do not declare the element as abstract.

XML-24061: element *string* not nillable

Cause: [cvc-elt.3.1] There was an attribute `xsi:nil`, which was not allowed because the element declaration was not nillable.

Action: Remove `xsi:nil` attribute from the element.

XML-24062: no character or element children allowed for nil content *string*

Cause: [cvc-elt.3.2.1] Element was specified `nil` but had character or element children.

Action: Remove any element content or remove `nil` attribute.

XML-24063: nil element does not satisfy fixed value constraint

Cause: [cvc-elt.3.2.2] Element had an fixed value while the content in instance was empty.

Action: Remove `nil` attribute from element.

XML-24064: xsi:type not a QName at line *string*, column *string*

Cause: [cvc-elt.4.1] The value of attribute `xsi:type` was not a QName.

Action: Change the value to a valid QName that references a type.

XML-24065: xsi:type *string* not resolved to a type definition

Cause: [cvc-elt.4.2] The referenced type specified by `xsi:type` was absent.

Action: Correct the value of `xsi:type` so it points to a valid type definition.

XML-24066: local type *string* not validly derived from the type of element *string*

Cause: [cvc-elt.4.3] The type referenced by `xsi:type` was not derived from original type.

Action: Modify the reference type definition so that it satisfies the constraint, or use another type that is derived from the original type.

XML-24067: value *string* not in enumeration

Cause: [cvc-enumeration-valid] The value was not one in the enumeration constraint.

Action: Use valid value specified in enumeration.

XML-24068: invalid facet *string* for type *string*

Cause: [cvc-facet-valid] The given data value violates the constraining facet.

Action: Correct the data value.

XML-24069: too many fraction digits in value *string* at line *string*, column *string*

Cause: [cvc-fractionDigits-valid] The given number violated the `fractionDigits` constraining facet.

Action: Use fewer fraction digits.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-24070: missing ID definition for ID reference *string* at line *string*, column *string*

Cause: [cvc-id.1] There is no `ID` binding in the `ID/IDREF` table for validation root

Action: Define the `ID` for the `ID` reference

XML-24071: duplicate ID *string* at line *string*, column *string*

Cause: [cvc-id.2] Same `ID` was defined more than once.

Action: Eliminate duplicate `ID` attributes.

XML-24072: duplicate key sequence *string*

Cause: [cvc-identity-constraint] The document contained duplicate key sequence that violated uniqueness constraint.

Action: Correct the document to make key sequence unique, or modify XPath to avoid it.

XML-24073: target node set not equals to qualified node set for key *string*

Cause: [cvc-identity-constraint.4.2.1] There were empty key sequences in key constraint.

Action: Make sure every element in target node set has a nonempty key sequence.

XML-24074: element member *string* in key sequence is nillable

Cause: [cvc-identity-constraint.4.2.3] The element selected as a member in a key sequence was nillable, which is not allowed.

Action: Modify the schema to make corresponding element declaration not nillable.

XML-24075: missing key sequence for key reference *string*

Cause: [cvc-identity-constraint.4.3] A `keyref` referenced to empty key sequence.

Action: Make sure every key sequence for `keyref` is has a corresponding key sequence for referenced key.

XML-24076: incorrect length of value *string*

Cause: [cvc-length-valid] The length of the value was not the same as specified in length facet.

Action: Use data value with correct length.

XML-24077: value *string* greater than or equal to maxExclusive

Cause: [cvc-maxExclusive-valid] The data value was out of boundary specified in `maxExclusive` facet.

Action: Correct the data value.

XML-24078: value *string* greater than the maxInclusive

Cause: [cvc-maxInclusive-valid] The data value was out of boundary specified in `maxInclusive` facet.

Action: Correct the data value.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-24079: value length of *string* greater than maxLength

Cause: [cvc-maxLength-valid] The length of the data value was greater than `maxLength`.

Action: Make the data value's length smaller than `maxLength`.

XML-24080: value *string* smaller or equals to minExclusive

Cause: [cvc-minExclusive-valid] The data value was out of lower boundary of value range.

Action: Use data value that is greater to `minExclusive`.

XML-24081: value *string* smaller than minInclusive

Cause: [cvc-minInclusive-valid] The data value was too small.

Action: Use data value not smaller than the value of `minInclusive`.

XML-24082: value *string* shorter than minLength

Cause: [cvc-minLength-valid] The length of value was smaller than that specified in `minLength`.

Action: Use data value with length greater than or equals to `minLength`.

XML-24083: wildcard particle in the content of element *string* not done

Cause: [cvc-particle.1.1] The wildcard particle's `minOccurs` had not been met.

Action: Have more elements in the content that match the wildcard.

XML-24084: element particle *string* not done

Cause: [cvc-particle.1.2] The element particle's `minOccurs` had not been met.

Action: Have more elements that match the element declaration or members in its substitution group.

XML-24085: model group *string* in the content of element *string* not done

Cause: [cvc-particle.1.3] The model group particle's `minOccurs` had not been met.

Action: Have more elements in the content that match the model group.

XML-24086: invlid literal *string* with respect to pattern facet *string*

Cause: [cvc-pattern-valid] The literal did not match the pattern constraining facet.

Action: Correct the lexical data to match pattern facet.

XML-24087: undefined type *string*

Cause: [cvc-resolve-instance.1] Could not resolve the type reference to a type definition

Action: Add the type definition to schema

XML-24088: undeclared attribute *string*

Cause: [cvc-resolve-instance.2] Could not resolve attribute reference to an attribute declaration.

Action: Add the attribute declaration to schema.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-24089: undeclared element *string*

Cause: [cvc-resolve-instance.3] Could not resolve element reference to an element declaration

Action: Add the element declaration to schema.

XML-24090: undefined attribute group *string*

Cause: [cvc-resolve-instance.4] Could not resolve the attribute group reference to an attribute group definition.

Action: Define the attribute group definition in schema.

XML-24091: undefined model group *string*

Cause: [cvc-resolve-instance.5] Could not resolve the model group reference to a model group definition.

Action: Define the model group in schema.

XML-24092: undeclared notation *string*

Cause: [cvc-resolve-instance.6] Could not resolve the notation reference to a notation declaration.

Action: Add the notation declaration to schema.

XML-24093: too many digits in value *string* at line *string*, column *string*

Cause: [cvc-totalDigits-valid] The number of digits in numeric value was greater than the value of `totalDigits` facet.

Action: Use smaller numbers.

A.6 Schema Representation Constraint Error Messages

Schema representation constraint error messages are in the range XML-24100 through XML-24199.

XML-24100: element *string* must belong to XML Schema namespace

Cause: Element in XML Schema document did not have Schema namespace.

Action: Specify XML Schema namespace http://www.w3.org/2001/XMLSchema

XML-24101: can not build schema from location *string*

Cause: [schema_reference.2] Processor could not find schema from given schema location

Action: Fix the schema location

XML-24102: can not resolve schema by target namespace *string*

Cause: [schema_reference.3] Processor was unable to retrieve schema based on given namespace.

Action: Fix the schema namespace

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-24103: invalid annotation representation at line *string*, column *string*

Cause: [src-annotation]

XML-24104: multiple annotations at line *string*, column *string*

Cause: [src-annotation] More than one annotation elements appeared in component.

Action: Remove extra annotation.

XML-24105: annotation must be the first element at line *string*, column *string*

Cause: [src-annotation] Annotation was not the first element in component.

Action: Move annotation to the begining of component content.

XML-24106: attribute wildcard before attribute declaration at line *string*, column *string*

Cause: The attribute wildcard appeared before attribute declarations.

Action: Move attribute wildcard to the end of declaration.

XML-24107: multiple attribute wildcard

Cause: [src-attribute.1] More than one anyAttributes were declared.

Action: Remove extra attribute wildcards.

XML-24108: default *string* and fixed *string* both present

Cause: [src-attribute.1] Both default and fixed attriubtes were present in attriubte declaration.

Action: Remove either default or fixed attribute.

XML-24109: default value *string* conflicts with attribute use *string*

Cause: [src-attribute.2] Both default and use were present, and value for use is not optional.

Action: Remove either default or use value.

XML-24110: missing name or ref attribute

Cause: [src-attribute.3.1] Neither name nor ref attribute was present in declaration.

Action: Add name or ref to the declaration.

XML-24111: both name and ref presented in attribute declaration

Cause: [src-attribute.3.1] Name and ref attribute were both present in attribute declaration.

Action: Add name or ref to the declaration.

XML-24112: ref conflicits with form, type, or simpleType child

Cause: [src-attribute.3.2] The attribute was a reference, and form, type or simpleType child were specified.

Action: Either change ref to name, or remove form; or remove type, or children, or both.

XML-24113: type attribute conflicts with simpleType child

Cause: [src-attribute.4] Both type attribute and simpleType child were present.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Action: Remove either type reference or type definition.

XML-24114: intersecton of attribute wildcard is not expressible

Cause: [src-attribute_group.2] Attriubes wildcards defined were not expressible with a wildcard.

Action: Remove inexpressible attribute wildcards.

XML-24115: circular attribute group reference *string*

Cause: [src-attribute_group.3] Attriubte group were circularly referenced outside redefine

Action: Remove circular reference

XML-24116: circular group reference *string*

Cause: group were circularly referenced outside redefine.

Action: Remove circular reference

XML-24117: base type *string* for complexContent is not complex type

Cause: [src-ct.1] Derived a complexType with complex content from simple type

Action: Change base type to complex type

XML-24118: simple content required in base type *string*

Cause: [src-ct.2] A complexType with simpleContent was derived from a complexType with complex content

Action: Change base type to simple type (if derivation is extension) or simpleContent complex type.

XML-24119: properties specified with element reference *string*

Cause: [src-element.2.2] Element reference also had complexType, simpleType, key, keyrefunique children or nillable, form, default, block, or type attribute.

Action: Remove conflict attributes or children.

XML-24120: simpleType and complexType can not both present in element declaration *string*

Cause: [src-element.3] Element declaration had both complexType, simpleType children.

Action: Remove either simpleType or complexType child.

XML-24121: imported namespace *string* must different from namespace *string*

Cause: [src-import.1.1] The namespce of import was the same as the target namespace of importing schema

Action: Change import to inclusion.

XML-24122: target namespace *string* required

Cause: [src-import.1.2] Imported namespace was specified but absent imported schema.

Action: Remove namespace attribute in element import, or add target namespac to the imported schema.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-24123: namespace *string*is different from expedted targetNamespace *string*

Cause: [src-import.3.1] Specified namespace was different from actual targetNamespace impported.

Action: Correct the namespace attribute in import element.

XML-24124: targetNamespace *string* not expected in schema

Cause: [src-import.3.2] Specified a no-namespace schema, but actual schema had targetNamespace.

Action: Remove the imported schema's targetNamespace attribute

XML-24125: can not include schema from *string*

Cause: [src-include.1] Processor was unable to include a schema from given location.

Action: Check correctness of URL and URL resolver

XML-24126: included targetNamespace *string* must the same as *string*

Cause: [src-include.2.1] Tried to include a achema with different targetNamespace.

Action: Use import instead of include.

XML-24127: no-namespace schema can not include schema with target namespace *string*

Cause: [src-include.2.2] A schema without targetNamespace tried to include a schema with targetNamespace.

Action: Use import instead of include

XML-24128: itemType attribute conflicits with simpleType child

Cause: [src-list-itemType-or-simpleType] Both itemType attribute and simpleType child were present in list simple type declaration.

Action: Remove either itemType attribute or simpleType child.

XML-24129: prefix of qname *string* can not be resolved

Cause: [src-qname] Prefix of a qname was present, but did not map to any in-scope namespace.

Action: Declare a namespace corresponding to the prefix.

XML-24130: redefined schema has different namespace. line *string* column *string*

Cause: Redefined schema's targetNamespace was not the same as the targetNamespace of redefining schema.

Action: Correct the targetNamespace in redefined schema.

XML-24131: no-namespace schema can only redefine schema without targetNamespace

Cause: [src-redefine.3.2] A no-namespace schema tried to redefine a schema with namespace

Action: Remove the targetNamespace attribute from redefined schema.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-24132: type derivation *string* must be restriction

Cause: [src-redefine.5] A simpleType or complexType was present in redefine, but the derivation was not restriction.

Action: Change the type redefinition, make it a restriction.

XML-24132: type *string* must redefine itself at line *string*, column *string*

Cause: [src-redefine.5] A simpleType or complexType was present in redefine, but its base type was not itself.

Action: Change the base type to redefine itself.

XML-24133: group *string* can have only one self reference in redefinition

Cause: [src-redefine.6.1.1] A group was present in redefine and it had more than onereferences to itself in its content.

Action: Remove extra self references in the group redefinition.

XML-24134: self reference of group *string* must not have minOccurs or maxOccurs other than 1 in redefinition

Cause: [src-redefine.6.1.2] A minOccurs or maxOccurs with value other than 1 was specified in a group self reference in redefine.

Action: Remove the minOccurs or maxOccurs attribute.

XML-24135: redefined group *string*is not a restriction of its orginal group

Cause: [src-redefine.6.2.2] A group presented in redefine, without self reference but was nota valid restriction of its original group.

Action: Modify the content of the group, make it a valid restriction of its original.

XML-24236: attribute group *string* can have only one self reference in redefinition

Cause: [src-redefine.7.1] An attributeGroup was present in redefine and it had more than oneself references in its content.

Action: Remove extra self references.

XML-24136: redefined attribute group *string* must be a restriction of its orginal group

Cause: [src-redefine.7.2.2] An attributeGroup presented in redefine, without self reference but was not a valid restriction of its original.

Action: Modify the content of the attribute group, make it valid restriction of its original.

XML-24137: restriction must not have both base and simpleType child

Cause: [src-restriction-base-or-simpleType]

XML-24138: simple type restriction must have either base attribute or simpleType child

Cause: [src-simple-type.2] Both base and simpleType were absent in simple type restriction

Action: Add either base attribute or simpleType child.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-24139: neitehr itemType or simpleType child present for list

Cause: [src-simple-type.3] Missing itemType attribute or simpleType child in list definition.

Action: Add either itemType or simpleType child

XML-24140: itemType and simpleType child can not both be present in list type.

Cause: [src-simple-type.3] Both itemType attribute and simpleType child were present inlist definition

Action: Remove either itemType or simpleType child.

XML-24141: circular union type is disallowed

Cause: [src-simple-type.4] Some member types in union type made references to the union type

Action: Remove the circular references

XML-24142: facet *string* can not be specified more than once

Cause: [src-single-facet-value] Same facet other than enumeration and pattern had been specified more than once, which is not allowed.

Action: Remove extra facets.

XML-24143: memberTypes and simpleType child can not both be absent in union

Cause: [src-union-memberTypes-or-simpleTypes] Both memberTypes and simpleType were absent for a union type.

Action: Either specify memberTypes or add simpleType children.

XML-24144: facets can only used for restriction

Cause: [st-restrict-facets] Derivation was not restriction while facet children were present.

Action: Remove facet children.

A.7 Schema Component Constraint Error Messages

Schema component constraint error messages are in the range XML-24200 through XML-24399.

XML-24201: duplicate attribute *string* declaration

Cause: [ag-props-correct.1] There were more than one attribute declarations with same namespace and name in attribute group definition.

Action: Remove duplicate attribute declarations.

XML-24202: more than one attributes with ID type not allowed

Cause: [ag-props-correct.2] There were more than one attribute declarations with type ID.

Action: Change to other types for such attriubte declarations

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-24203: invalid value constraint *string*

Cause: [a-props-correct.2] The fixed value or default value did not satisfy the attribute's type

Action: Use type valid default for fixed value.

XML-24204: value constraint *string* not allowed for ID type

Cause: [a-props-correct.3] Attribute with ID type had either fixed or default value constraint.

Action: Remove value constraint.

XML-24205: fixed value *string* does not match *string* in attribute declaration

Cause: [au-props-correct.2] Attriubte reference specified a fixed value which is not the same as that in referenced declaration.

Action: Correct the fixed value to the same as specified in attribute declaration

XML-24206: value constraint must be fixed to match that in attribute declaration

Cause: [au-props-correct.2] Attriubte reference specified a default value, while the referenced declaration had a fixed value.

Action: Remove default value form attribute reference.

XML-24207: invalid xpath expression *string*

Cause: [c-fields-xpaths.1] The value of xpath was not valid xpath expression as specified in XPath 1.0.

Action: Use correct xpath

XML-24208: invalid field xpath *string*

Cause: [c-fields-xpaths.2] The value of xpath did not satisfied field's restricted xpath syntax.

Action: Correct the xpath expression

XML-24209: maxOccurs in element *string* of All group must be 0 or 1

Cause: [cos-all-limited] Some elements in a All group had maxOccurs greater than one.

XML-24210: All group has to form a content type.

Cause: All group was contained in another model group

Action: Make all group at the top of a content type

XML-24211: All group has to form a content type.

Cause: [cos-applicable-facets] All group was contained in another model group

Action: Make all group at the top of a content type

XML-24212: type *string* does not allow facet *string*

Cause: [cos-applicable-facets] A facet not applicable to the simple type was used.

Action: Remove the facet.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-24213: wildcard intersection is not exprssible

Cause: [cos-aw-intersect] Two wilcards in an attribtue group had different negative namespaces

Action: Use only one wildcard with negative namespace

XML-24214: base type not allow *string* derivation

Cause: [cos-ct-derived-ok.1] Base type's final prevented the derivation.

Action: Remove the derivation method from the value of final in base type

XML-24215: complex type *string* is not a derivation of type *string*

Cause: [cos-ct-derived-ok.2] There was no derivation chain from base type to derived type.

Action: Fix the derivation chaining.

XML-24216: must specify a particle in extened content type

Cause: [cos-ct-extends.1.4.2.1] The content type of an extension of a complex type was empty

Action: Add particle to the content type of extension.

XML-24217: content type *string* conflicts with base type's content type *string*

Cause: [cos-ct-extends.1.4.2.2.2] Base type's content type was not empty and was not the same as the content type specified.

Action: Match the specified content type with that in base type.

XML-24218: inconsistent local element declarations *string*

Cause: More than one elements in the content had same name and namespace, but did not refer to the same type.

Action: Make type references the same for all elements equal in name and namespace

Comments: cos-element-consistent

XML-24219: element *string* is not valid substitutable for element *string*

Cause: [cos-equiv-derived-ok-rec]

XML-24220: itemType *string* can not be list

Cause: [cos-list-of-atomic] The itemType of a list type was itself a list.

Action: Use atomic or union type as the itemType of list.

XML-24221: cricular union *string* not allowed

Cause: [cos-no-circular-union] Union's name and namespace matched one of its memberType.

Action: Remove any circular references

XML-24222: ambiguous particles *string*

Cause: [cos-nanambig] particles in a content type violated UPA (Unique Particle Attrition) constraint.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Action: Make content type particle unambiguous.

XML-24223: invalid particle extension

Cause: [cos-particle-extend]

XML-24224: invalid particle restriction

Cause: [cos-particle-restrict]

XML-24225: simple type *string* does not allowed restriction

Cause: [cos-st-derived-ok] Derivation was restriction but restriction was in base type's final.

Action: Remove restriction from base type's final.

XML-24226: invalid derivation from base type *string*

Cause: [cos-st-derived-ok] The derivation violated the "type derivaton OK (simple)" constraint.

Action: Make the derivation satisfy the constraint.

XML-24227: atomic type can not restrict list *string*

Cause: [cos-st-restricts.1.1] base type is list,

XML-24228: base type can not be ur-type in restriction

Cause: [cos-st-restricts.1.1] Tried to directly restrict anySimipleType.

XML-24229: base type of list must be list or ur-type

Cause: [cos-st-restricts.2.3]

XML-24230: base type of union must be union or ur-type

Cause: [cos-st-restricts.3.3]

XML-24231: element default *string*requires mixed content to be emptiable

Cause: [cos-valide-default] Element had default constraint but its mixed content type was not emtiable.

Action: Remove default value constraint.

XML-24232: element default *string* requires mixed content or simple content

Cause: [cos-valide-default] Element had default value constraint but its content type was element only or empty.

Action: Remove default value constraint.

XML-24233: element default *string* must be valid to its content type

Cause: [cos-valide-default] Element's default value constraint was invalid to its type.

Action: Correct the default value or remove it.

XML-24234: wrong field cardinality for keyref *string*

Cause: [c-props-correct] Number of fields were different between keyref and referenced key.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Action: Ensure that keyref and referenced key have same number of fields.

XML-24235: complex type can only extend simple type *string*

Cause: [ct-props-correct] Complex type was derived from simple type, but derivation was not extension.

Action: Change restriction to extension.

XML-24236: cricular type definition *string*

Cause: [ct-props-correct] Type was in its own derivation chain.

Action: Remove recursive derivation.

XML-24237: base type *string* must be complex type

Cause: [derivation-ok-restriction.1] Complex type was restricted from a simple type.

Action: Change the restriction from a complex type.

XML-24238: attribute *string* not allowed in base type

Cause: [derivation-ok-restriction.2] The attribute in restriction was not allowed for base type.

Action: Correct the restriction of attribute use.

XML-24239: required attribute *string* not in restriction

Cause: [derivation-ok-restriction.3] Restriction's attribute uses was not a subset of basetype's attribute uses.

Action: Correct the restriction of attribute uses.

XML-24240: no correspoonding attribue wildcard in bas type *string*

Cause: [derivation-ok-restriction.4] Restriction had an attribute wildard that did not corrspond to any attribute wildcard in base type.

Action: Correct the derivation.

XML-24241: base type *string* must have simple content or emptiable

Cause: [derivation-ok-restriction.5.1] Content type was simple, but the base type has complex content that is not mixed or not emptiable.

Action: Change the content type from simple to element only.

XML-24242: base type *string* must have empty content or emptiable

Cause: [derivation-ok-restriction.5.2] Content type was empty, but the base type had simple content or not emptiable complex content.

Action: Change the content type from simple to element only.

XML-24243: enumeration facet required for NOTATION

Cause: [enumeration-required-notation] NOTATION type was used without enumeration facet.

Action: Specify enumeration facet for NOTATION.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-24244: invalid value *string* in enumeration

Cause: [enumeration-valid-restriction] Some value in enumeration was not valid in respect to the type.

Action: Correct invalid values.

XML-24245: default value *string*is element type invalid

Cause: [e-props-correct.2] Default value was invalid in respect to the type of element.

Action: Correct the default value.

XML-24246: invalid substitutionGroup *string*, type invalid

Cause: [e-props-correc.3] The type of the element was not a validly derivation from the type ofelement's substitutionGroup.

Action: Correct the type or remove substitutionGroup.

XML-24247: ID type does not allow value constraint *string*

Cause: [e-props-correct.4] Type was ID or its derivation whild there was a value constraint.

Action: Remove value constraint.

XML-24248: fractionDigits *string*greater than totalDigits *string*

Cause: [fractionDigits-totalDigits] The value for fractionDigits was greater than totalDigits.

Action: Make fractionDigits smaller or equal to totalDigits.

XML-24249: length facet can not be specified with minLength or maxLength

Cause: [length-minLength-maxLength] Both length and either minLength or maxLength were specified.

Action: Remove length facet.

XML-24250: length *string* not the same as length in base type's

Cause: [length-valid-restriction] Specified a length that was not the same as the length in base type.

Action: Remove length facet.

XML-24251: maxExclusive greater than its original

Cause: [maxExclusive-valid-restriction] Restricted maxExclusive was greater thant its original in base type.

XML-24252: minInclusive greater than or equal to maxExclusive

Cause: [maxInclusive-maxExclusive] Specified a minInclusive that was greater or equal to maxExclusive.

Action: Make minInclusive smaller than maxExclusive.

XML-24253: maxLength is greater than that in base type

Cause: [maxLength-valid-restriction] Specified a maxLength greater than orginal in base type.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Action: Sepcify a smaller maxLength to make it valid restriction.

XML-24254: circular group *string*disallowed

Cause: [mg-props-correct] Circular model group references.

Action: Remove circular references in model group definition.

XML-24256: minExclusive must be less than or equal to maxExclusive

Cause: [minExclusive-less-than-equals-to-maxExclusive] minExclusive was bigger than maxExclusive.

Action: Use smaller value for minExclusive.

XML-24257: minExclusive *string*must be less than maxInclusive

Cause: [minExclusive-less-than-maxInclusive] inExclusive specified was greater than or equal to maxInclusive.

Action: Specify smaller minExclusive.

XML-24258: invalid minExclusive *string*

Cause: [minExclusive-valid-restriction] Restriction's minExclusive was less than base type's minExclusive

Action: Specify greater value for minExclusive.

XML-24259: invalid minExclusive *string*

Cause: [minExclusive-valid-restriction] Restriction's minExclusive was less than base type's minInclusive

Action: Specify greater value for minExclusive

XML-24260: invalid minExclusive *string*

Cause: [minExclusive-valid-restriction] Restriction's minExlcusive was greater than base type's maxInclusive

Action: Specify smaller value for minExclusive

XML-24261: invalid minExclusive *string*

Cause: [minExclusive-valid-restriction] Restriction's minExclusive was greater than or equals to base type's maxExclusive

Action: Specify smaller value for minExclusive.

XML-24262: minInclusive *string* must not be greater than maxInclusive

Cause: [minInclusive-less-than-equal-to-maxInclusive] Specified a minInclusive that was greater than maxInclusive

Action: Specify smaller value for minInclusive.

XML-24263: Can not specify both minInclusive and minExclusive

Cause: [minInclusive-minExclusive]] Restriction specified both minInclusive and minExclusive.

Action: Remove either minInclusive or minExclusive.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-24264: invalid minInclusive *string*

Cause: [minInclusive-valid-restriction] Restriction's minInclusive was less than or equal to minInclusive in base type.

Action: Use minInclusive larger than that of base type.

XML-24265: invalid minInclusive *string*

Cause: [minInclusive-valid-restriction] Restriction's minInclusive was less than minExclusivein base type.

Action: Use minInclusive larger than or equal to the minExclusive of base type.

XML-24267: invalid minInclusive *string*

Cause: [minInclusive-valid-restriction] Restriction's minInclusive was greater than maxInclusive in base type.

Action: Use minInclusive smaller than or equal to the maxInclusive of base type.

XML-24268: invalid minInclusive *string*

Cause: Restriction's minInclusive was greater than or equal to maxEnclusive in base type.

Action: Use minInclusive smaller than the maxEnclusive of base type.

Comments: minInclusive-valid-restriction

XML-24269: invalid minLength *string*

Cause: [minLength-less-than-equal-to-maxLength] minLength in restriction is greater than base type's maxLength.

Action: Make minLength within the length range of base type.

XML-24270: invalid minLength *string*

Cause: [minLength-valid-restriction] Value of minLength is smaller than that of base type in restriction.

Action: Use bigger value for minLength.

XML-24271: can not declare xmlns attribute

Cause: [no-xmlns] Declared an attribute with name xmlns.

Action: Remove such declaraton.

XML-24272: no xsi for targetNamespace

Cause: [no-xsi] The schema's target namespace matched http://www.w3.org/2001/XMLSchema-instance

Action: Use other target namespace.

XML-24272: minOccurs is greater than maxOccurs

Cause: [n-props-correct] The minOccurs of particle was greater than the maxOccurs.

Action: Use smaller value for minOccurs.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-24281: maxOccurs must greater than or equal to 1

Cause: [p-props-correct] The maxOccurs of particle was less than 1.

Action: Use greater value for maxOccurs.

XML-24282: incorrect Notation properties

Cause: [n-props-correct] The Notation declaration had incorrect properties.

Action: Fix Noation declaration.

XML-24283: particle's range is not valid restriction

Cause: [range-ok] Range of restriction was not within the range of parent particle.

XML-24284: sequence group is not valid derivation of choice group

Cause: Restriction did not satisfy constraint: Particle Derivation OK (Sequence:Choice -- MapAndSum)

Comments: rcase-MapAndSum

XML-24285: element *string* is not valid restriction of element *string*

Cause: [rcase-NameAndTypeOK] Restriction did not satisfy constraint: Particle Restriction OK

XML-24286: element *string* is not valid restriction of wildcard

Cause: [rcase-NSCompat] Restriction did not satisfy constraint: Particle Restriction OK

XML-24287: group is not valid restriction of wildcard

Cause: [rcase-NSRecurseCheckCardinality] Restriction did not satisfy constraint: Particle Restriction OK

XML-24288: group any is not valid restriction

Cause: [rcase-NSSubset] Restriction did not satisfy constraint: Particle Restriction OK(Any:Any -- NSSubset)

XML-24289: invalid restriction of all or sequence group

Cause: [rcase-Recurse] Restriction did not satisfy constraint: Particle Restriction OK(All:All, Seqiemce"Sequence:-- Recurse)

XML-24290: wildcard is not valid restriction

Cause: [rcase-RecurseLax] The wildcard was not validly restricted from another wildcard.

XML-24291: sequence is not a valid restriction of all

Cause: Restriction violated constraint: Particle Derivation OK (Sequence:All--RecurseUnordered)

Action: Fix the restriction.

XML-24292: duplicate component definitions *string*

Cause: [sch-props-correct] There were two schema components with same name and namespace.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Action: Remove duplicate definitions.

XML-24293: Incorrect simple type definition properties

Cause: [st-props-correct]

XML-24294: wildcard is not a subset of its super

Cause: [w-props-correct] The namespace constraint was not a restriction of its super

Action: Correct namespace constraint.

XML-24295: totalDigits *string*is greater than *string* in base type

Cause: [totalDigits-valid-restriction] Restriction specified a totalDigits with value greater than that in base type.

Action: Use smaller value for totalDigits.

XML-24296: whiteSpace *string* can not restrict base type's *string*

Cause: [whiteSpace-valid-restriction] Restriction's whiteSpace was replace or preserve, and base had whiteSpace collapse, or restriction had replace while base had preserve.

Action: Eliminate conflicit whiteSpace values.

XML-24297: circular substitution group *string*

Cause: Substitution group was circular.

Action: Remove the circular substitution group

Copyright © 1999, 2025, Oracle and/or its affiliates.

A.8 XSQL Server Pages Error Messages

XSQL server error messages are in the range XML-25000 through XML-25999.

XML-25001: Cannot locate requested XSQL file. Check the name.

XML-25002: Cannot acquire database connection from pool: *string*

XML-25003: Failed to find config file *string* in CLASSPATH.

XML-25004: Could not acquire a database connection named: *string*

XML-25005: XSQL page is not well-formed.

XML-25006: XSLT stylesheet is not well-formed: *string*

XML-25007: Cannot acquire a database connection to process page.

XML-25008: Cannot find XSLT Stylesheet: *string*

XML-25009: Missing arguments on command line

XML-25010: Error creating: *string*\nUsing standard output.

XML-25011: Error processing XSLT stylesheet: *string*

XML-25012: Cannot Read XSQL Page

XML-25013: XSQL Page URI is null; check exact case of file name.

XML-25014: Resulting page is an empty document or had multiple document elements.

XML-25015: Error inserting XML Document

XML-25016: Error parsing posted XML Document

XML-25017: Unexpected Error Occurred

XML-25018: Unexpected Error Occurred processing stylesheet *string*

XML-25019: Unexpected Error Occurred reading stylesheet *string*

XML-25020: Config file *string* is not well-formed.

XML-25021: Serializer *string* is not defined in XSQL configuration file

XML-25022: Cannot load serializer class *string*

XML-25023: Class *string* is not an XSQL Serializer

XML-25024: Attempted to get response Writer after getting OutputStream

XML-25025: Attempted to get response OutputStream after getting Writer

XML-25026: Stylesheet URL references an untrusted server.

XML-25027: Failed to load *string* class for built-in xsql:*string* action.

XML-25028: Error reading *string*. Check case of the name.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Action: None required

XML-30001: Error occurred in execution of Process

Cause: Component being wrapped by pipeline process is causing error

Action: Fix input XML content

XML-30002: Only XML type(s) *string* allowed.

Comments: Must not occur normally

XML-30003: Error creating/writing to output *string*

Cause: Output URL provided might be invalid

XML-30004: Error creating base url *string*

Cause: URL provided as base URL is invalid

Action: Fix base URL provided

XML-30005: Error reading input *string*

Cause: Input URL provided might be invalid

XML-30006: Error in processing pipedoc Error element

XML-30007: Error converting output to xml type required by dependent process

XML-30008: A valid parameter target is required

Cause: Parameter with name target is missing or invalid

Action: Add parameter target pointing to the target output label

XML-30009: Error piping output to input

XML-30010: Process definition element *string* needs to be defined

Cause: Element `procdef` is missing

Action: Add process definition to `pipedoc`.

XML-30011: ContentHandler not available

Cause: The dependent process does not provide a valid `ContentHandler`

Action: Implement the `getContentHandler` API in your Process.

XML-30012: Pipeline components are not compatible

Cause: Component output and input don't match in terms of document/docfrag.

Action: Fix the `pipedoc` to use components which are compatible.

XML-30013: Process with output label *string* not found

Cause: Process whose output label matched target label is not available.

Action: Create a process in the `pipedoc`, where the output label matches the label of the target parameter.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-30014: Pipeline is not complete, missing output/outparam label called *string*

Cause: A dependent process output label has not been named correctly, or a dependent process is missing.

Action: Ensure that each dependent input has a corresponding output.

XML-30016: Unable to instantiate class

Cause: A process could not be created because there is an error in the process definition element associated with it.

Action: Correctly specify the class for a process definition.

XML-30017: Target is up-to-date, pipeline not executed

Cause: Either the target does not exist, or the pipeline inputs are more recent than the target.

Action: Use the `force` option to execute pipeline regardless of whether the target is up to date.

A.10 JAXB Error Messages

Java Architecture for XML Binding (JAXB) error messages are in the range XML-32000 through XML32999.

XML-32202: a problem was encountered because multiple <schemaBindings> were defined.

Cause: There was more than one instance of `<schemaBindings>` declaration in the annotation element of the `<schema>` element.

Action: Update the annotation to remove duplicate `<schemaBinding>` declaration.

XML-32203: a problem was encountered because multiple <class> name annotations were defined on node *string*.

Cause: There was more than one instance of `<class>` declaration in the annotation element of the node.

Action: Update the annotation to remove duplicate `<class>` declaration.

XML-32204: a problem was encountered because the name in <class> declaration contained a package name prefix *string* which was not allowed.

Cause: A failure occurred because the name attribute in the `<class>` declaration contained a package prefix.

Action: Update the `className` in the `<class>` declaration.

Comments: The package prefix is inherited from the current value of package.

XML-32205: a problem was encountered because the property customization was not specified correctly on node *string*.

Cause: A failure occurred because the property customization was not specified correctly.

Action: Update the `<property>` customization.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML-32206: a problem was encountered because the javaType customization was not specified correctly on node *string*.

Cause: A failure occurred because the property customization was not specified correctly.

Action: Update the `<javaType>` customization.

XML-32207: a problem was encountered in declaring the baseType customization on the node *string*.

Cause: A failure occurred because the `<baseType>` customization was not specified correctly.

Action: Update the `<baseType>` customization.

XML-32208: a problem was encountered because multiple baseType customizations were declared on the node *string*.

Cause: A failure occurred because multiple `<baseType>` customizations were declared.

Action: Remove one of the `<baseType>` customization declaration.

XML-32209: a problem was encountered because multiple javaType customizations were declared on the node *string*.

Cause: A failure occurred because multiple `<javaType>` customizations were declared.

Action: Remove one of the `<javaType>` customization declaration.

XML-32210: a problem was encountered because invalid value was specified on customization of *string*.

Cause: A failure occurred because an invalid value was specified on the `globalBindings` customization declaration.

Action: Check and correct the `globalBindings` customization value.

XML-32211: a problem was encountered because incorrect <schemaBindings> customization was specified.

Cause: A failure occurred because an invalid value was specified on the `schemaBindings` customization.

Action: Check and correct the `schemaBindings` customization value.

XML-32212: the <class> customization did not support specifiying the implementation class using implClass declaration. The implClass declaration specified on node *string* was ignored.

Cause: A warning occurred because the `implClass` customization declaration was not supported.

XML-32213: the <globalBindings> customization did not support specifiying user specific class that implements java.util.List. The collectionType declaration was ignored.

Cause: A warning occurred because the user specific implementation class for `java.util.List` was not supported.

Copyright © 1999, 2025, Oracle and/or its affiliates.

B

XDK for Java TXU Error Messages

Error messages are listed for applications that use Oracle XML Developer's Kit (XDK) for Java during the execution of TXU interfaces.

B.1 DLF Error Messages

Data Loading Format (DLF) error messages are in the range TXU-0100 through TXU-0199.

TXU-0100: parameter *string* in query *string* not found

Cause: There is not a placeholder for the parameter in the query

Action: Supply a parameter whose id can be found as an associated placeholder in the associated query

TXU-0101: incompatible attributes col and constant coexist at *string* in query *string*

Cause: Attributes 'col' and 'constant' cannot coexist

Action: Remove either 'col' or 'constant' attribute

TXU-0102: node *string* not found

Cause: The document lacks an expected node

Action: Supply the missing node

TXU-0103: element *string* lacks content

Cause: The element has no data

Action: Supply content

TXU-0104: element *string* with SQL *string* lacks col or constant attribute

Cause: The element lacks a required attribute of 'col' or 'constant'

Action: Supply either 'col' or 'constant' attribute

TXU-0105: SQL exception *string* while processing SQL *string*

Cause: An error occurred during the SQL execution

Action: Resolve the error in the SQL statement

TXU-0106: no data for column *string* selected by SQL *string*

Cause: The SQL query returned no data

Action: Supply data or modify your query

Copyright © 1999, 2025, Oracle and/or its affiliates.

TXU-0107: datatype *string* not supported

Cause: An attempt to process an unsupported data type was made

Action: Change the data type to a supported one

TXU-0108: missing maxsize attribute for column *string*

Cause: The size-unit attribute is specified but maxsize is not.

Action: Supply the maxsize attribute, too

TXU-0109: a text length of *string* for *string* exceeds the allowed maximum of *string*

Cause: The length of the text data is too long

Action: Shorten the data so it fits in the limit, or enlarge the maxsize attribute and ensure the database column is large enough

TXU-0110: undeclared column *string* in row *string*

Cause: A column in the data section is not declared in the columns section

Action: Modify the column name to a declared one

TXU-0111: lacking column data for *string* in row *string*

Cause: A column is declared but the data is missing.

Action: Supply the col element whose name attribute matches the column name

TXU-0112: undeclared query parameter *string* for column *string*

Cause: The query parameter refers to an undeclared column

Action: Specify a declared column

TXU-0113: incompatible attribute *string* with a query on column *string*

Cause: A column with a query cannot have the specified attribute

Action: Remove either the attribute or query

TXU-0114: DLF parse error (*string*) on line *string*, character *string* in *string*

Cause: The format is in error as reported

Action: Correct the erroneous part

TXU-0115: The specified date string *string* has an invalid format

Cause: The specified date string does not match the specified formatstring.

Action: Make sure the date string is in an appropriate date format

Copyright © 1999, 2025, Oracle and/or its affiliates.

B.2 TransX Informational Messages

TransX informational messages are in the range TXU-0200 throughTXU-0299.

TXU-0200: duplicate row at *string*

Cause: A duplicate row exists in the database

Action: This message appears on the DuplicateRowException to inform applications of existance of one or more duplicate rows already stored in the database

B.3 TransX Error Messages

TransX error messages are in the range TXU-0300 through TXU-0399.

TXU-0300: document *string* not found

Cause: The document could not be located

Action: Modify the document location or supply the document at the location

TXU-0301: file *string* could not be read

Cause: An I/O error happened when reading the file

Action: Resolve the I/O problem

TXU-0302: archive *string* not found

Cause: The archive file could not be located

Action: Ensure that the CLASSPATH includes TransX correctly and only once

TXU-0303: schema *string* not found in *string*

Cause: The schema definition of DLF could not be located

Action: Get an unbroken copy of a TransX archive

TXU-0304: archive path for *string* not found

Cause: The path for the archive could not be determined

Action: Ensure that the CLASSPATH includes TransX correctly and only once

TXU-0305: no database connection on *string* call for *string*

Cause: The operation was attempted without a database connection

Action: Open a connection first

TXU-0306: null tablename given; access denied

Cause: The table name is not provided

Action: Specify a table name

Copyright © 1999, 2025, Oracle and/or its affiliates.

TXU-0307: lookup-keys could not be determined *string*

Cause: The data dictionary is corrupted

Action: Restore the data dictionary

TXU-0308: binary file *string* not found

Cause: The file name is invalid

Action: Supply a good file name

TXU-0309: a file size of *string* exceeds the allowed maximum of 2,000 bytes

Cause: The file is too large

Action: Reduce the file size

B.4 Assertion Error Messages

Assertion error messages are in the range TXU-0400 through TXU-0499.

TXU-0400: missing SQL statement element on *string*

Cause: An internal assertion was not successful

Action: Contact Oracle customer support

TXU-0401: missing node *string*

Cause: An internal assertion was not successful

Action: Contact Oracle customer support

TXU-0402: invalid flag *string*

Cause: An internal assertion was not successful

Action: Contact Oracle customer support

TXU-0403: internal error *string*

Cause: An internal assertion was not successful

Action: Contact Oracle customer support

TXU-0404: unexpected Exception *string*

Cause: An internal assertion was not successful

Action: Contact Oracle customer support

Copyright © 1999, 2025, Oracle and/or its affiliates.

C

XDK for Java XSU Error Messages

Error messages are listed for applications that use Oracle XML Developer's Kit (XDK) for Java during the execution of the XML SQL Utility (XSU) interfaces.

C.1 Generic Error Messages

Generic error messages are in the range XSUE-0000 through XSUE-0099.

XSUE-0000: Internal Error -- Exception Caught *string*

XSUE-0001: Internal Error -- *string*

XSUE-0002: *string* is not a scalar column. The row id attribute can only get values from scalar columns.

XSUE-0003: *string* is not a valid column name.

XSUE-0004: This object has been closed. If you would like the object not to be closed implicitly between calls, see the *string* method.

XSUE-0005: The row-set enclosing tag and the row enclosing tag are both omitted; consequently, the result can consist of at most one row which contains exactly one column which is not marked to be an XML attribute.

XSUE-0006: The row enclosing tag or the row-set enclosing tag is ommitted; consequently to get a well formed XML document, the result can only consist of a single

Copyright © 1999, 2025, Oracle and/or its affiliates.

row with multiple columns or multiple rows with exactly one column which is not marked to be an XML attribute.

XSUE-0007: Parsing of the sqlname failed -- invalid arguments.

XSUE-0008: Character *string* is not allowed in an XML tag name.

XSUE-0009: this method is not supported by *string* class. Please use *string* instead.

XSUE-0010: The number of bind names does not equal the number of bind values.

XSUE-0011: The number of bind values does not match the number of binds in the SQL statement.

XSUE-0012: Bind name identifier *string* does not exist in the sql query.

XSUE-0013: The bind identifier has to be of non-zero length.

XSUE-0014: Root node supplied is null.

XSUE-0015: Invalid LOB locator specified.

XSUE-0016: File *string* does not exit.

XSUE-0017: Can not create oracle.sql.STRUCT object of a type other than oracle.sql.STRUCT (i.e. ADT).

XSUE-0018: Null is not a valid DocumentHandler.

XSUE-0019: Null and empty string are not valid namespace aliases.

XSUE-0020: to use this method you will have to override it in your subclass.

XSUE-0021: You are using an old version of the gss library; thus, sql-xml name escaping is not supported.

XSUE-0022: cannot create XMLType object from opaque base type: *string*

C.2 Query Error Messages

Query error messages are in the range XSUE-0100 through XSUE-0199.

XSUE-0100: Invalid context handle specified.

XSUE-0101: In the FIRST row of the resultset there is a nested cursor whose parent cursor is empty; when this condition occurs we are unable to generate a dtd.

XSUE-0102: *string* is not a valid IANA encoding.

XSUE-0103: The resultset is a "TYPE_FORWARD_ONLY" (non-scrollable); consequently, xsu can not reposition the read point. Furthermore, since the result set has been passed to the xsu by the caller, the xsu can not recreate the resultset.

XSUE-0104: input character is invalid for well-formed XML: *string*

C.3 DML Error Messages

Data manipulation language (DML) error messages are in the range XSUE-0200 through

Copyright © 1999, 2025, Oracle and/or its affiliates.

XSUE-0299.

XSUE-0200: The XML element tag *string* does not match the name of any of the columns/attributes of the target database object.

XSUE-0201: NULL is an invalid column name.

XSUE-0202: Column *string*, specified to be a key column, does not not exits in table *string*.

XSUE-0203: Column *string*, specified as column to be updated, does not exist in the table *string*.

XSUE-0204: Invalid REF element - *string* - attribute *string* missing.

XSUE-0206: Must specify key values before calling update routine. Use the *string* function.

XSUE-0207: UpdateXML: No columns to update. The XML document must contain some non-key columns to update.

XSUE-0208: The key column array must be non empty.

XSUE-0209: The key column array must be non empty.

XSUE-0210: No rows to modify -- the row enclosing tag missing. Specify the correct row enclosing tag.

XSUE-0211: *string* encountered during processing ROW element *string* in the XML document.

XSUE-0212: *string* XML rows were successfully processed.

XSUE-0213: All prior XML row changes were rolled back.

Copyright © 1999, 2025, Oracle and/or its affiliates.

D

Oracle XML Developer's Kit JavaBeans (Deprecated)

A description is given of Oracle XML Developer's Kit (XDK) JavaBeans, which is deprecated.

> **Note**
>
> The XDK JavaBeans, described in this appendix, and the corresponding XDK Java application programming interface (API) packages and classes are deprecated in Oracle Database 12c Release 1 (12.1). These components are still supported in Oracle Database 12c Release 1 (12.1), but Oracle recommends not using them in new applications. This functionality is deprecated with no replacement.

> **See Also**
>
> *Oracle AI Database XML Java API Reference* for more information about the deprecated XDK Java APIs

D.1 Introduction to XDK JavaBeans

XDK JavaBeans are a set of Extensible Markup Language (XML) components that you can use in Java applications and applets.

D.1.1 Prerequisites for Using XDK JavaBeans

Prerequisites for using XDK JavaBeans are described.

This appendix assumes that you are familiar with these technologies:

- JavaBeans. JavaBeans components, or Beans, are reusable software components that can be manipulated visually in a builder tool.
- Java Database Connectivity (JDBC). Database connectivity is included with the XDK JavaBeans. The beans can connect directly to a JDBC-enabled database to retrieve and store XML and Extensible Stylesheet Language (XSL) files.
- Document Object Model (DOM). DOM is an in-memory tree representation of the structure of an XML document.
- Simple API for XML (SAX). SAX is a standard for event-based XML parsing.
- XML Schema language. See Using the XML Schema Processor for Java for an overview and links to suggested reading.

Copyright © 1999, 2025, Oracle and/or its affiliates.

D.1.2 Standards and Specifications for XDK JavaBeans

XDK JavaBeans require version 1.2 or later of XDK, and they can be used with any version of JDK 1.2 or above. The XDK JavaBeans conform with the Sun JavaBeans specification, and include the required `BeanInfo` class that extends `java.beans.SimpleBeanInfo`.

The JavaBeans 1.01 specification describes JavaBeans as present in JDK 1.1.

The additions for the Java 2 platform to the JavaBeans core specification provide developers with standard means to create more sophisticated JavaBeans components.

Related Topics

- Oracle XML Developer's Kit Standards
  A description is given of the Oracle XML Developer's Kit (XDK) standards.

> **See Also**
>
> XDK on OTN for the JavaBeans specifications

D.1.3 XDK JavaBeans Features

XDK JavaBeans facilitate the addition of graphical user interfaces (GUIs) to XML applications. Bean encapsulation includes documentation and descriptors that you can access directly from Java integrated development environments (IDEs) such as Oracle JDeveloper.

D.1.3.1 DOMBuilder

The `oracle.xml.async.DOMBuilder` bean constructs a DOM tree from an XML document. The `DOMBuilder` JavaBean encapsulates the XML parser for class `DOMParser` with a bean interface and supports asynchronous parsing. By registering a listener, Java programs can initiate parsing of large or multiple documents and immediately return control to the caller.

A main benefit of this bean is increased efficiency when parsing multiple files, especially if the files are large. `DOMBuilder` can also provide asynchronous parsing in a background thread in interactive visual applications. Without asynchronous parsing, the GUI is useless until the document to be parsed. With `DOMBuilder`, the application invokes the parse method and then resumes control. The application can display a progress bar, allow the user to cancel the parse, and so forth.

Related Topics

- Using the DOMBuilder JavaBean: Basic Process
  Class `DOMBuilder` implements an XML 1.0 parser according to the World Wide Web Consortium (W3C) recommendation. It parses an XML document and builds a DOM tree. The parsing is done in a separate thread. Interface `DOMBuilderListener` must be used for notification when the tree is built.

D.1.3.2 XSLTransformer

The `oracle.xml.async.XSLTransformer` JavaBean supports asynchronous transformation. It accepts an XML document, applies an Extensible Stylesheet Language Transformation (XSLT)

Copyright © 1999, 2025, Oracle and/or its affiliates.

stylesheet, and creates an output file. It lets you transform an XML document to almost any text-based format, including XML, HTML, and data definition language (DDL).

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 main benefit of the `XSLTransformer` bean is that it can transform multiple files in parallel. Like `DOMBuilder`, you can also use it in visual applications to avoid long periods of time when the GUI is nonresponsive. By implementing the `XSLTransformerListener` interface, the invoking application receives notification when the transformation completes.

Related Topics

- Using the XSLTransformer JavaBean: Basic Process
  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.

D.1.3.3 DBAccess

The `oracle.xml.dbaccess.DBAccess` bean maintains character large object (`CLOB`) tables that contain multiple XML and text documents.

You can use it to store and retrieve XML documents in the database, but not to process them within the database. Java applications that use the `DBAccess` bean connect to the database through JDBC. XML documents stored in `CLOB` tables that are not of type `XMLType` do not have their entities expanded.

The `DBAccess` bean enables you to do perform these tasks:

- Create and delete tables of type `CLOB`.
- Query the contents of `CLOB` tables.
- Perform `INSERT`, `UPDATE`, and `DELETE` operations on XML documents stored in `CLOB` tables.

D.1.3.4 XMLDBAccess

The `oracle.xml.xmldbaccess.XMLDBAccess` bean extends the `DBAccess` bean to support XML documents stored in `XMLType` tables. The class provides methods to list, delete, or retrieve `XMLType` instances and their tables. For example, the `getXMLXPathTextData()` method retrieves the value of an XPath expression from an XML document.

`DBAccess` JavaBean maintains `XMLType` tables that can hold multiple XML and text documents. Each XML or text document is stored as a row in the table. The table is created with this structured query language (SQL) statement:

```
CREATE TABLE (FILENAME   CHAR( ) UNIQUE,
              FILEDATA   SYS.XMLType);
```

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.

The `XMLDBAccess` bean performs these tasks:

- Creates and deletes `XMLType` tables

Copyright © 1999, 2025, Oracle and/or its affiliates.

- Lists the contents of an `XMLType` column
- Performs `INSERT`, `UPDATE`, and `DELETE` operations on XML documents stored in `XMLType` tables

Related Topics

- Using the XMLDBAccess JavaBean: Basic Process
  Basic use of the `XMLDBAccess` bean is described.

D.1.3.5 XMLDiff

When comparing XML documents, it is usually unhelpful to compare them character by character. Most XML comparisons are concerned with differences in structure and significant textual content, not differences in white space.

The `oracle.xml.differ.XMLDiff` bean performs these tasks:

- Constructs and compares the DOM trees for two input XML documents and indicates whether the documents are different.
- Provides a graphical display of the differences between two XML files. Specifically, you can refer to node insert, delete, modify, or move.
- Generates an XSLT stylesheet that can convert one of the input XML documents into the other document.

The `XMLDiff` bean is especially useful in pipeline applications. For example, an application could update an XML document, compare it with a previous version of the document, and store the XSLT stylesheet that shows the differences between them.

Related Topics

- Using the XML Pipeline Processor for Java
  An explanation is given of how to use the Extensible Markup Language (XML) pipeline processor for Java.
- Using the XMLDiff JavaBean: Basic Process
  Basic use of the `XMLDiff` bean is described.

D.1.3.6 XMLCompress

The Oracle XML parser includes a compressor that can serialize XML document objects as binary streams.

This is explained in Compressing and Decompressing XML. Although a useful tool, compression with XML parser has these disadvantages:

- When XML data is deserialized, it must be reparsed.
- The encapsulation of XML data in tags greatly increase its size.

The `oracle.xml.xmlcomp.XMLCompress` bean is an encapsulation of the XML compression functionality. It provides these advantages when serializing and deserializing XML:

- It encapsulates the method that serializes the DOM, which produces a stream.
- XML processors can regenerate the DOM from the compressed stream without reparsing the XML.

The bean supports compression and decompression of input XML parsed by `DOMParser` or `SAXParser`. DOM compression supports inputs of type `XMLType`, `CLOB`, and `BLOB`.

Copyright © 1999, 2025, Oracle and/or its affiliates.

To use different parsing options, parse the document before input and then pass the `XMLDocument` object to the compressor bean. The compression factor is a rough value based on the file size of the input XML file and the compressed file. The limitation of the compression factor method is that it can be used only when the compression is performed with `java.io.File` objects as parameters.

D.1.3.7 XSDValidator

The `oracle.xml.schemavalidator.XSDValidator` bean encapsulates the `XSDValidator` class and adds capabilities for validating a DOM tree.

A useful feature of this bean concerns validation errors. If the application throws a validation error, method `getStackList()` returns a list of DOM tree paths that lead to the invalid node. Nodes with errors are returned in a vector of stack trees in which the top element of the stack represents the root node. You can get child nodes by pulling them from the stack. To use `getStackList()` you must use instantiate the `java.util.Vector` and `java.util.Stack` classes.

D.2 Using XDK JavaBeans: Overview

Topics here include basic use of JavaBeans and running the demo programs.

D.2.1 Using XDK JavaBeans: Basic Process

The program flow of Java applications that use the more useful beans is described. These include `DOMBuilder`, `XSLTransformer`, `XMLDBAccess`, and `XMLDiff`.

D.2.1.1 Using the DOMBuilder JavaBean: Basic Process

Class `DOMBuilder` implements an XML 1.0 parser according to the World Wide Web Consortium (W3C) recommendation. It parses an XML document and builds a DOM tree. The parsing is done in a separate thread. Interface `DOMBuilderListener` must be used for notification when the tree is built.

When developing applications that use this bean, you must import these subpackages:

- `oracle.xml.async`, which provides asynchronous Java beans for DOM building
- `oracle.xml.parser.v2`, which provides APIs for SAX, DOM, and XSLT

Subpackage `oracle.xml.parser.v2` is described in XML Parsing for Java. The most important DOM-related classes and interfaces in the `javax.xml.async` package are described in Table D-1.

Table D-1 javax.xml.async DOM-Related Classes and Interfaces

Class/Interface	Description
`DOMBuilder` class	Encapsulates an XML parser to parse an XML document and build a DOM tree. The parsing is done in a separate thread. The `DOMBuilderListener` interface must be used for notification when the tree is built.
`DOMBuilderEvent` class	Instantiates the event object that `DOMBuilder` uses to notify all registered listeners about parse events.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table D-1 (Cont.) javax.xml.async DOM-Related Classes and Interfaces

Class/Interface	Description
`DOMBuilderListener` interface	Must be implemented so that the program can receive notifications about events during the asynchronous parsing. The class implementing this interface must be added to the `DOMBuilder` with the `addDOMBuilderListener()` method.
`DOMBuildeErrorEvent` class	Defines the error event that is sent when parse exception occurs.
`DOMBuilderErrorListener` interface	Must be implemented so that the program can receive notifications when errors are found during parsing. The class implementing this interface must be added to the `DOMBuilder` with the `addDOMBuilderErrorListener()` method.

Figure D-1 depicts the basic process of an application that uses the `DOMBuilder` JavaBean.

Figure D-1 DOMBuilder JavaBean Usage

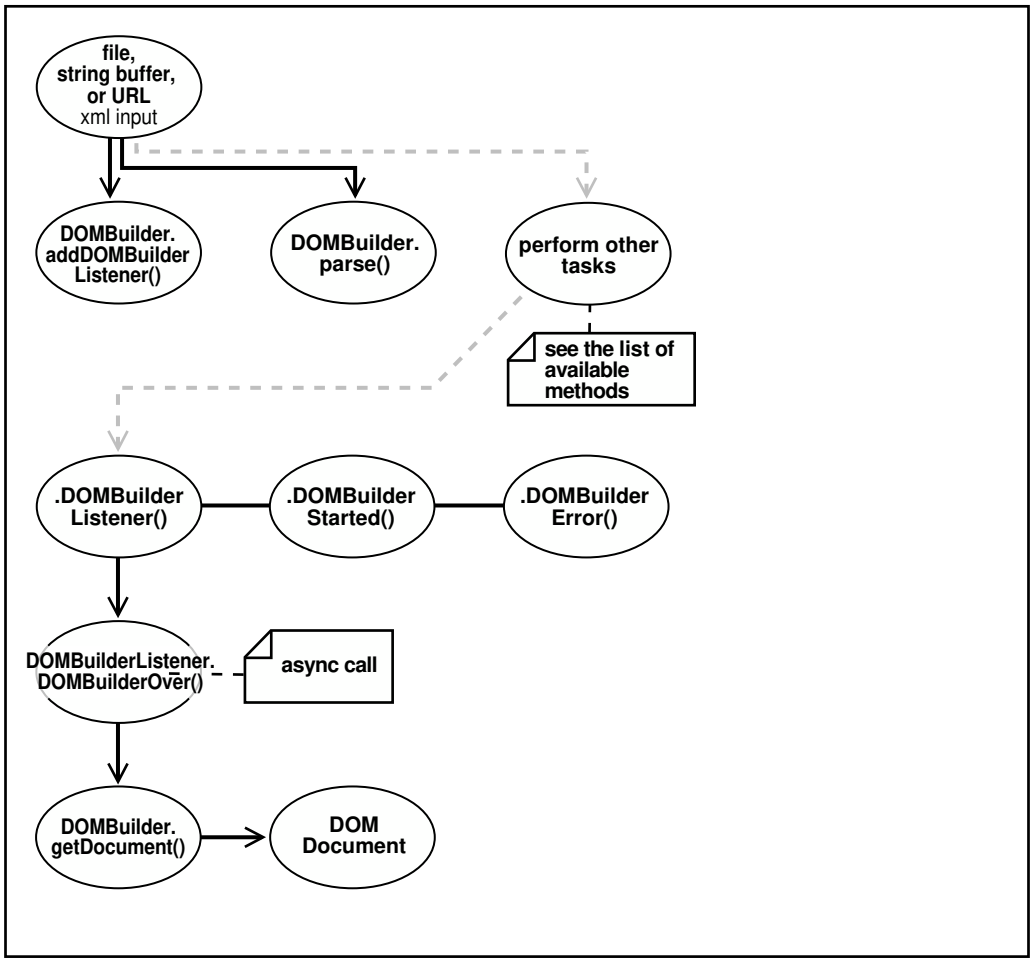


Figure D-1 shows these stages of XML processing:

1. Parse the input XML document. The program can receive the XML document as a file, string buffer, or URL.
2. Add the `DOMBuilder` listener. The program invokes the method `DOMBuilder.addDOMBuilderListener(DOMBuilderListener)`.

Copyright © 1999, 2025, Oracle and/or its affiliates.

3. Parse the XML document. The program invokes the `DOMBuilder.parse()` method.
4. Optionally, process the parsed result further.
5. Invoke the listener when the program receives an asynchronous call. The listener, which must implement the `DOMBuilderListener` interface, is called by invoking the `DOMBuilderOver()` method.

   The available `DOMBuilderListener` methods are:

   - `domBuilderError(DOMBuilderEvent)`, which is called when parse errors occur.
   - `domBuilderOver(DOMBuilderEvent)`, which is called when parsing completes.
   - `domBuilderStarted(DOMBuilderEvent)`, which is called when parsing begins.
6. Fetch the DOM. Invoke the `DOMBuilder.getDocument()` method to fetch the resulting DOM document and output it.

D.2.1.2 Using the XSLTransformer JavaBean: Basic Process

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.

When developing applications that use this bean, you must import these subpackages:

- `oracle.xml.async`, which provides asynchronous Java beans for XSL transformations
- `oracle.xml.parser.v2`, which provides APIs for XML parsing SAX, DOM, and XSLT

The `oracle.xml.parser.v2 subpackage` is described in detail in XML Parsing for Java. The most important XSL-related classes and interfaces in the `javax.xml.async` package are described in Table D-2.

Table D-2 javax.xml.async XSL-Related Classes and Interfaces

Class/Interface	Description
`XSLTransformer` class	Applies XSL transformation in a background thread.
`XSLTransformerEvent` class	Represents the event object used by `XSLTransformer` to notify XSL transformation events to all of its registered listeners.
`XSLTransformerListener` interface	Must be implemented so that the program can receive notifications about events during asynchronous transformation. The class implementing this interface must be added to the `XSLTransformer` with the `addXSLTransformerListener()` method.
`XSLTransformerErrorEvent` class	Instantiates the error event object that `XSLTransformer` uses to notify all registered listeners about transformation error events.
`XSLTransformerErrorListener` interface	Must be implemented so that the program can receive notifications about error events during the asynchronous transformation. The class implementing this interface must be added to the `XSLTransformer` using `addXSLTransformerListener()` method.

Figure D-2 shows `XSLTransformer` bean usage.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Figure D-2 XSLTransformer JavaBean Usage

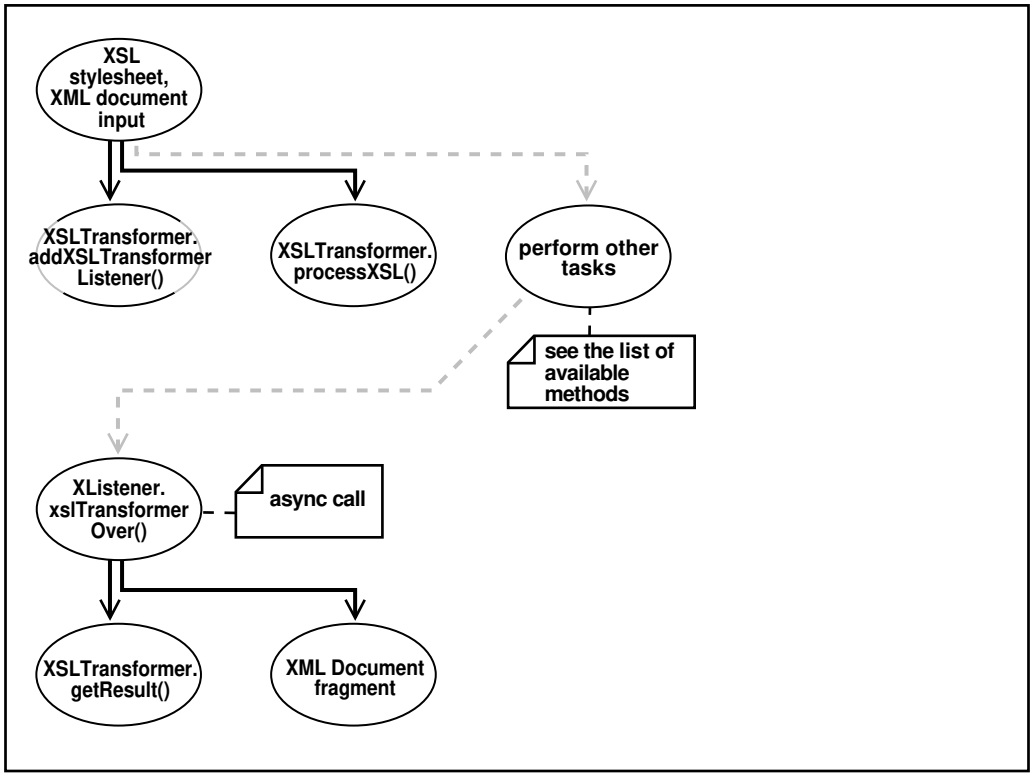


Figure D-2 goes through these stages:

1. Input an XSLT stylesheet and XML instance document.
2. Add an XSLT listener. The program invokes the `XSLTransfomer.addXSLTransformerListener()method`.
3. Apply the stylesheets. The `XSLTransfomer.processXSL()` method initiates the XSL transformation in the background.
4. Optionally, perform further processing with the `XSLTransformer` bean.
5. Invoke the XSLT listener when the program receives an asynchronous call. The listener, which must implement the `XSLTransformerListener` interface, is called by invoking the `xslTransformerOver()` method.
6. Fetch the result of the transformation. Invoke the `XSLTransformer.getResult()` method to return the XML document fragment for the resulting document.
7. Output the XML document fragment.

D.2.1.3 Using the XMLDBAccess JavaBean: Basic Process

Basic use of the `XMLDBAccess` bean is described.

When developing applications that use the `XMLDBAccess` bean, you must use these subpackages:

- `oracle.xml.xmldbaccess`, which includes the `XMLDBAccess` bean
- `oracle.xml.parser.v2`, which provides APIs for XML parsing SAX, DOM, and XSLT

Copyright © 1999, 2025, Oracle and/or its affiliates.

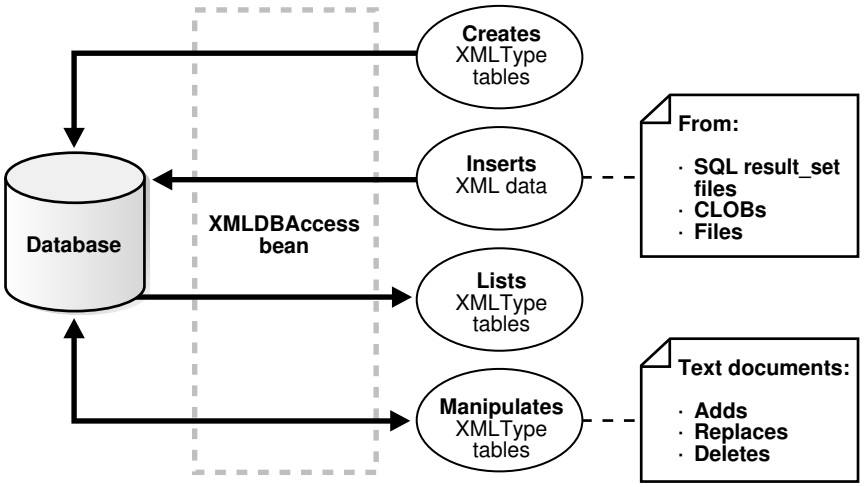
The `oracle.xml.parser.v2` subpackage is described in detail in XML Parsing for Java. Some of the more important methods in the `XMLDBAccess` class are described in Table D-3.

Table D-3 XMLDBAccess Methods

Class/Interface	Description
`createXMLTypeTable()`	Creates an `XMLType` table.
`insertXMLTypeData()`	Inserts a text file as a row in an `XMLType` table.
`replaceXMLTypeData()`	Replaces a text file as a row in an `XMLType` table.
`getXMLTypeTableNames()`	Retrieves all XML tables with names starting with a specified string.
`getXMLTypeData()`	Retrieves text file from an `XMLType` table.
`getXMLTypeXPathTextData()`	Retrieves the text data based on the XPath expression from an `XMLType` table.

Figure D-3 shows typical `XMLDBAccess` bean usage. It shows how the `DBAccess` bean maintains and manipulates XML documents stored in `XMLTypes`.

Figure D-3 XMLDBAccess JavaBean Usage



For example, an `XMLDBAaccess` program could process XML documents in these stages:

1. Create an `XMLType` table. Invoke `createXMLTypeTable()` by passing it database connection information and a table name.
2. List the `XMLType` tables. Invoke `getXMLTypeTableNames()` by passing it database connection information and an empty string.
3. Load XML data into the table. Invoke `replaceXMLTypeData()` by passing it database connection information, the table name, the XML file name, and a string containing the XML.
4. Retrieve the XML data into a `String`. Invoke `getXMLTypeData()` by passing it the connection information, the table name, and the XML file name.
5. Retrieve XML data based on an XPath expression. Invoke `getXMLXPathTextData()` by passing it the connection information, the table name, the XML file name, and the XPath expression.

Copyright © 1999, 2025, Oracle and/or its affiliates.

6. Close the connection.

D.2.1.4 Using the XMLDiff JavaBean: Basic Process

Basic use of the `XMLDiff` bean is described.

When developing applications that use the `XMLDiff` bean, you typically use these subpackages:

- `oracle.xml.xmldiff`, which includes the `XMLDiff` bean
- `oracle.xml.parser.v2`, which provides APIs for XML parsing SAX, DOM, and XSLT
- `oracle.xml.async`, which provides asynchronous Java beans for DOM building

Subpackage `oracle.xml.parser.v2` is described in detail in XML Parsing for Java. Some important methods in class `XMLDiff` are described in Table D-4.

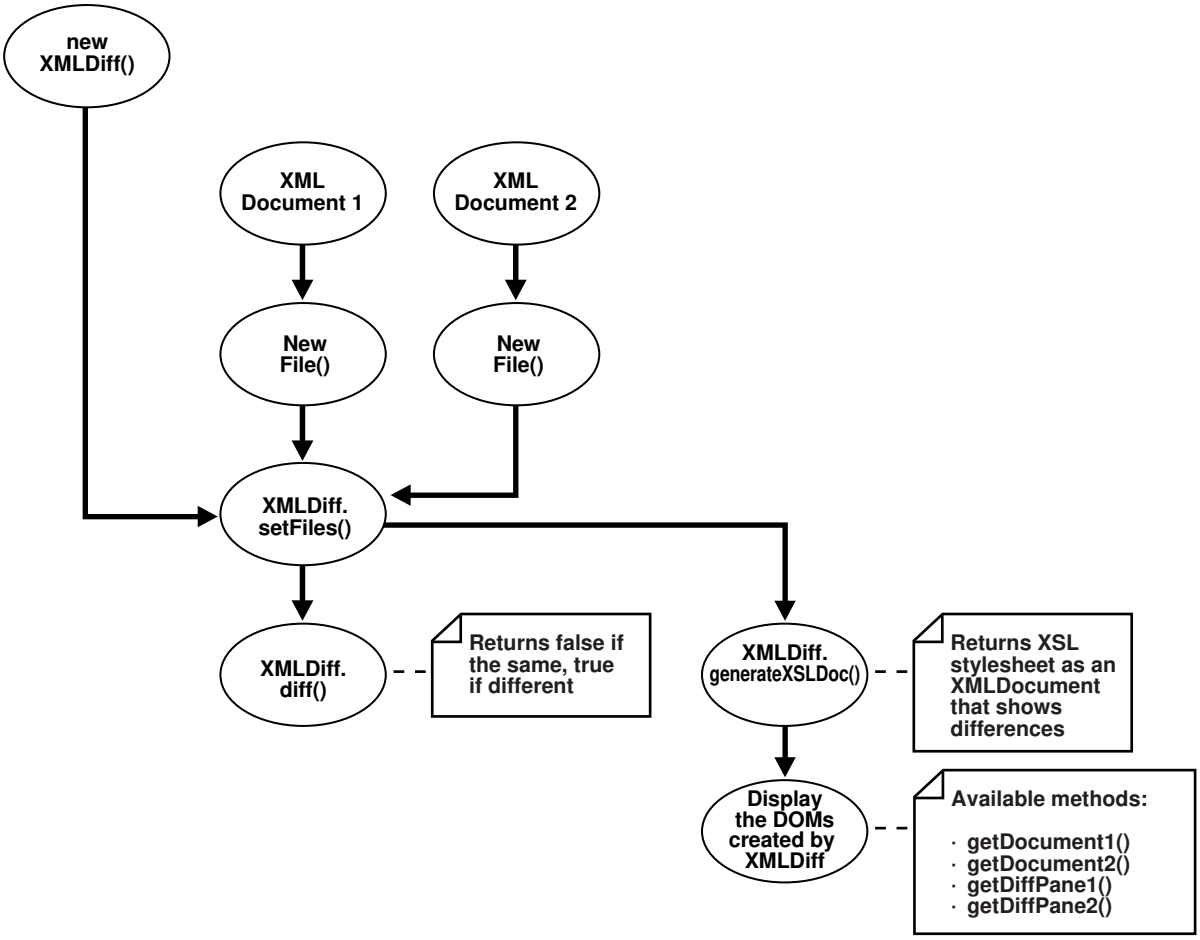
Table D-4 XMLDiff Methods

Class/Interface	Description
`diff()`	Determines the differences between two input XML files or two XMLDocument objects.
`generateXSL()`	Generates an XSL file that represents the differences between the input two XML files. The first XML file can be transformed into the second XML file with the generated stylesheet. If the XML files are the same, then the XSL generated can transform the first XML file into the second XML file, where the first and second files are equivalent. Related methods are `generateXSLDoc()` and `generateXSLFile()`.
`setFiles()`	Sets the XML files to be compared.
`getDocument1()`	Gets the document root as an `XMLDocument` object of the first XML tree. `getDocument2()` is the equivalent method for the second tree.
`getDiffPane1()`	Gets the text panel as `JTextPane` object that visually shows the diffs in the first XML file. `getDiffPane2()` is the equivalent method for the second file.

Figure D-4 shows typical `XMLDiff` bean usage. It shows how `XMLDiff` bean compares and displays the differences between input XML documents.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Figure D-4 XMLDiff JavaBean Usage



For example, an `XMLDiff` program could process XML documents in these stages:

1. Create an `XMLDiff` object.
2. Set the files to be compared. Create `File` objects for the input files and pass references to the objects to `XMLDiff.setFiles()`.
3. Compare the documents. The `diff()` method returns `false` if the XML files are the same and `true` if they are different.
4. Respond depending on the whether the input XML documents are the same or different. For example, if they are the same, invoke `JOptionPane.showMessageDialog()` to print a message.
5. Generate an XSLT stylesheet that shows the differences between the input XML documents. For example, `generateXSLDoc()` generates an XSL stylesheet as an `XMLDocument`.
6. Display the DOM trees created by `XMLDiff`.

Copyright © 1999, 2025, Oracle and/or its affiliates.

D.2.2 Running XDK JavaBean Demo Programs

Demo programs for XDK SJavaBeans are included in directory `$ORACLE_HOME/xdk/demo/java/transviewer`.

The demos show the use of the XDK beans described in XDK JavaBeans Features, and also some visual beans that are now deprecated. The beans used in the demos are:

- `XSLTransformer`
- `DOMBuilder`
- `DBAccess`
- `XMLDBAccess`
- `XMLDiff`
- `XMLCompress`
- `XSDValidator`
- `oracle.xml.srcviewer.XMLSourceView` (deprecated)
- `oracle.xml.treeviewer.XMLTreeView` (deprecated)
- `oracle.xml.transformer.XMLTransformPanel` (deprecated)
- `oracle.xml.dbviewer.DBViewer` (deprecated)

Although the visual beans are deprecated, they remain useful as educational tools. Consequently, the deprecated beans are included in `$ORACLE_HOME/lib/xmldemo.jar`. The nondeprecated beans are included in `$ORACLE_HOME/lib/xml.jar`.

Table D-5 lists the sample programs provided in the demo directory. The first column of the table indicates which sample program use deprecated beans.

Table D-5 JavaBean Sample Java Source Files

Sample	File Name	Description
`sample1` (deprecated)	`XMLTransformPanelSample.java`	A visual application that uses the `XMLTransformPanel`, `DOMBuilder`, and `XSLTransformer` beans. This bean applies XSL transformations to XML documents and shows the result.
`sample2` (deprecated)	`ViewSample.java`	A sample visual application that uses the `XMLSourceView` and `XMLTreeView` beans. It visualizes XML document files.
`sample3`	`AsyncTransformSample.java`	A nonvisual application that uses the `XSLTransformer` and `DOMBuilder` beans. It applies the XSLT stylesheet specified in `doc.xsl` on all `.xml` files in the current directory. It writes the results to files with the extension `.log`.
`sample4` (deprecated)	`DBViewSample.java`	A visual application that uses the `DBViewer` bean to implement a simple application that handles insurance claims.
`sample4` (deprecated)	`DBViewClaims.java`	This `JFrame` subclass is instantiated in the `DBViewFrame` class, which is in turn instantiated in the `DBViewSample` program.
`sample4` (deprecated)	`DBViewFrame.java`	This `JFrame` subclass is instantiated in the `DBViewSample` program.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table D-5 (Cont.) JavaBean Sample Java Source Files

Sample	File Name	Description
sample5	XMLDBAccessSample.java	A nonvisual application for the XMLDBAccess bean. This program demonstrates how to use the XMLDBAccess bean APIs to store and retrieve XML documents in XMLType tables. To use XMLType, you need Oracle Database and xdb.jar. The program accepts values for HOSTNAME, PORT, SID, USERID, and PASSWORD. The program creates tables in the database and loads data from file booklist.xml. The program writes output to xmldbaccess.log.
sample6 (deprecated)	XMLDiffSample.java	A visual application that uses the XMLDiff bean to find differences between two XML files and generate an XSLT stylesheet. You can use this stylesheet to transform the first input XML into the second input XML file.
sample6 (deprecated)	XMLDiffFrame.java	A class that implements the ActionListener interface. This class is used by the XMLDiffSample program.
sample6 (deprecated)	XMLDiffSrcView.java	A JPanel subclass used by the XMLDiffSample program.
sample7 (deprecated)	compviewer.java	A visual application that uses the XMLCompress bean to compress XML. The XML input can be an XML file or XML data obtained through a SQL query. The application enables you to decompress the compressed stream and view the resulting DOM tree.
sample7 (deprecated)	compstreamdata.java	A simple class that pipes information from the GUI to the bean. This class is used in dbpanel.java, filepanel.java, and xmlcompressutil.java.
sample7 (deprecated)	dbpanel.java	A JPanel subclass used in xmlcompressutil.java.
sample7 (deprecated)	filepanel.java	A JPanel subclass used in xmlcompressutil.java.
sample7 (deprecated)	xmlcompressutil.java	A JPanel subclass used in compviewer.java.
sample8 (deprecated)	XMLSchemaTreeViewer.java	A visual application that uses the Treeviewer, sourceviewer, and XSDValidator beans. The application accepts an XML instance document and an XML schema document as inputs. The application parses both the documents and validates the instance document against the schema. If the document is invalid, then the nodes where the errors occurred are highlighted and an error message is shown in a tool tip.
sample8 (deprecated)	TreeViewerValidate.java	A JPanel subclass that displays a parsed XML instance document as a tree. This class is used by the XMLSchemaTreeViewer.java program.
sample9 (deprecated)	XMLSrcViewer.java	A visual application that uses the sourceviewer and XSDValidator beans. The demo takes an XML file as input. You can select the validation mode: document type definition (DTD), XML schema, or no validation. The program validates the XML data file against the DTD or schema and displays it with syntax highlighting. It also logs validation errors. For schema validation it also highlights the error nodes appropriately. External and internal DTDs can be viewed.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Table D-5 (Cont.) JavaBean Sample Java Source Files

Sample	File Name	Description
sample9 (deprecated)	XMLSrcViewPanel.java	A class that shows how to use the XMLSourceView and DTDSourceView objects. This class is used by the XMLSrcViewer.java program.Each XMLSourceView object is set as a Component of a JPanel by invoking goButton_actionPerformed(). The XML file to be viewed is parsed and the resulting XML document is set in the XMLSourceView object by invoking makeSrcPane(). The highlighting and DTD display properties are specified at this time. For performing schema validation, build the schema object by invoking makeSchemaValPane(). You can can check for errors and display the source code accordingly with different highlights. You can retrieve a list of schema validation errors from the XMLSourceView by invoking dumpErrors().
sample10	XSDValidatorSample.java	An application that shows how to use the XSDValidator bean. It accepts an XML file and an XML schema file as input. The program displays errors occurring during validation, including line numbers.

Table D-6 describes additional files that are used by the demo programs.

Table D-6 JavaBean Sample Files

File Name	Description
XMLDiffData1.txt	An XML document used by the XMLDiffSample.java program. By default the 2 XML files XMLDiffData1.txt and XMLDiffData2.txt are compared and the output XSLT is stored as XMLDiffSample.xsl.
XMLDiffData2.txt	An XML document used by the XMLDiffSample.java program. By default the 2 XML files XMLDiffData1.txt and XMLDiffData2.txt are compared and the output XSLT is stored as XMLDiffSample.xsl.
booklist.xml	An XML document for use by XMLDBAccessSample.java.
claim.sql	An XML document used by ViewSample.java and XMLDBAccessSample.java.
doc.xml	An XML document for use by AsyncTransformSample.java.
doc.xsl	An XSLT stylesheet for use by AsyncTransformSample.java.
emptable.xsl	An XSLT stylesheet for use by AsyncTransformSample.java, ViewSample.java, or XMLTransformPanelSample.java.
note_in_dtd.xml	A sample XML document for use in XMLSrcViewer.java. You can use this file in DTD validation mode to view an internal DTD with validation errors. An internal DTD can be optionally displayed along with the XML data.
purchaseorder.xml	An XML document used by the XSDValidatorSample.java program. The instance document purchaseorder.xml does not conform to the XML schema defined in purchaseorder.xsd, which causes the program to display the errors.
purchaseorder.xsd	An XML schema document used by the XSDValidatorSample.java program. The instance document purchaseorder.xml does not conform to the XML schema defined in purchaseorder.xsd, which causes the program to display the errors.

Documentation for how to compile and run the sample programs is located in the README in the same directory. The basic steps are:

Copyright © 1999, 2025, Oracle and/or its affiliates.

1. Change into the `$ORACLE_HOME/xdk/demo/java/transviewer` directory (UNIX) or `%ORACLE_HOME%\xdk\demo\java\transviewer` directory (Windows).
2. Make sure that your environment variables are set as described in Setting Up the XDK for Java Environment. The beans require Java Development Kit (JDK) 1.2 or later. The `DBViewer` and `DBTransformPanel` beans require JDK 1.2.2 when rendering HTML. Prior versions of the JDK may not render HTML in the result buffer properly.
3. Edit the `Makefile` (UNIX) or `Make.bat` (Windows) for your environment. In particular,:
   - Change the `JDKPATH` in the `Makefile` to point to your JDK path.
   - Change `PATHSEP` to the appropriate path separator for your operating system.
   - Change the `HOSTNAME`, `PORT`, `SID`, `USERID`, and `PASSWORD` parameters so that you can connect to the database through the JDBC thin driver. These parameters are used in `sample4` and `sample5`.
4. Run `make` (UNIX) or `Make.bat` (Windows) at the system prompt to generate the class files.
5. Run `gmake` to run the demos:

```
gmake sample1
gmake sample2
gmake sample3
gmake sample4
gmake sample5
gmake sample6
gmake sample7
gmake sample8
gmake sample9
gmake sample10
```

D.2.2.1 Running sample1

`Sample1` is the program that uses the `XMLTransViewer` bean.

You can run the program manually:

```
java XMLTransformPanelSample
```

You can use the program to import and export XML files from Oracle Database, store XSL transformation files in the database, and apply stylesheets to XML interactively. To use the database connectivity feature in this program, you must know the network name of the computer where the database runs, the port (usually `1521`), and the name of the Oracle instance (usually `orcl`). You also need an account with `CREATE TABLE` privileges. If you have installed the sample schemas, then you can use the account `hr`. You can the `XMLTransViewer` program to apply stylesheet transformation to XML files and display the result.The program displays a panel with tabs on the top and the bottom. You can use the first two top tabs to switch between the XML buffer and the XSLT buffer. The third tab performs XSL transformation on the XML buffer and displays the result. You can use the first two tabs on the bottom to load and save data from Oracle Database and from the file system. The remaining bottom tabs switch the display of the current content to tree view, XML source, edit mode and, in case of the result view after the transformation, HTML.

Copyright © 1999, 2025, Oracle and/or its affiliates.

D.2.2.2 Running sample2

`Sample2` is a GUI-based demo for the `XMLSourceView` and `XMLTreeView` beans, which are deprecated. The `ViewSample` program displays the `booklist.xml` file in separate source and tree views.

You can run the program manually:

```
java ViewSample
```

D.2.2.3 Running sample3

`Sample3` is a nonvisual demo for the asynchronous `DOMBuilder` and `XSLTransformer` beans. The `AsyncTransformSample` program applies the `doc.xsl` XSLT stylesheet to all `*.xml` files in the current directory. The program writes output to files with the extension `.log`.

You can run the program manually:

```
java AsyncTransformSample
```

D.2.2.4 Running sample4

`Sample4` is a visual demo for the `DBViewer` bean, which is deprecated.

It runs in these stages:

1. It starts SQL*Plus, connects to the database with the `USERID` and `PASSWORD` specified in the `Makefile`, and runs the `claim.sql` script. This script creates several tables, views, and types for use by the `DBViewSample` demo program.
2. It runs the DBViewSample program:

   ```
   java -classpath "$(MAKE_CLASSPATH)" DBViewSample
   ```

JDBC connection information is hard-coded in the `DBViewClaims.java` source file, which implements a class used by the demo. Specifically, the program assumes the values for `USERID`, `PASSWORD`, and so forth set in the `Makefile`. If your configuration is different, navigate to line 92 in `DBViewClaims.java` and modify `setUsername()`, `setPassword()`, and so forth with values that reflect your Oracle Database configuration.

D.2.2.5 Running sample5

`Sample5` is a nonvisual demo for the `XMLDBAccess` bean. It uses the XMLType objects to store XML documents inside the database. The following program connects to the database with the Java thin client, creates `XMLType` tables, and loads the data from `booklist.xml`.

To run the program you must specify these pieces of information as command-line arguments:

- Host name (for example, `myhost`)
- Port number (for example, `1521`)
- SID of the database (for example, `ORCL`)
- Database account in which the tables are created (for example, `hr`)
- Password for the database account (for example, `hr`)

For example, you can run the program:

Copyright © 1999, 2025, Oracle and/or its affiliates.

```
java XMLDBAccessSample myhost 1521 ORCL hr hr
```

The following text shows sample output from `dbaccess.log`:

```
Demo for createXMLTypeTables():
Table +'testxmltype' successfully created.

Demo for listXMLTypeTables():
tablenamename=TESTXMLTYPE

Demo for replaceXMLTypeData() (similar to insert):
XML Data from +'booklist.xml' successfully replaced in table 'testxmltype'.

Demo for getXMLTypeData():
XMLType data fetched:
<?xml version="1.0"?>
<booklist>
  <book isbn="1234-123456-1234">
    <title>C Programming Language</title>
    <author>Kernighan and Ritchie</author>
    <publisher>EEE</publisher>
    <price>7.99</price>
  </book>
...
  <book isbn="1230-23498-2349879">
    <title>Emperor's New Mind</title>
    <author>Roger Penrose</author>
    <publisher>Oxford Publishing Company</publisher>
    <price>15.99</price>
  </book>
</booklist>

Demo for getXMLTypeXPathTextData():
Data fetched using XPath exp '/booklist/book[3]':
<book isbn="2137-598354-65978">
  <title>Twelve Red Herrings</title>
  <author>Jeffrey Archer</author>
  <publisher>Harper Collins</publisher>
  <price>12.95</price>
</book>
```

D.2.2.6 Running sample6

The `sample6` program is a visual demo for the `XMLDiff` bean.

The `XMLDiffSample` class invokes a GUI that enables you to choose the input data files from the **File** menu by selecting **Compare XML File**s. The **Transform** menu enables you to apply the generated XSLT generated to the first input XML. Select **Save As** in the **File** menu to save the output XML file, which is the same as the second input file. By default, the program compares `XMLDiffData1.txt` to `XMLDiffData2.txt` and stores the XSLT output as `XMLDiffSample.xsl`.

You can run the program manually:

```
java -mx50m XMLDiffSample XMLDiffData1.txt XMLDiffData2.txt
```

If the input XML files use a DTD that accesses a URL outside a firewall, then modify `XMLDiffSample.java` to include the proxy server settings before the `setFiles()` invocation. For example, modify the program as follows:

Copyright © 1999, 2025, Oracle and/or its affiliates.

```
/* Set proxy to access dtd through firewall */
Properties p = System.getProperties();
p.put("proxyHost", "www.proxyservername.com");
p.put("proxyPort", "80");
p.put("proxySet", "true");
/* You will also have to import java.util.*; */
```

D.2.2.7 Running sample7

The `sample7` visual demo shows the use of the `XMLCompress` bean. Class `compviewer` invokes a GUI that lets the user compress and uncompress XML files and data obtained from the database. The loading options let the user retrieve the data from a file system or a database.

This application does not support loading and saving compressed data from the database. The compression factor indicates a rough estimate by which the XML data is reduced.

You can run the program manually:

```
java compviewer
```

D.2.2.8 Running sample8

The `sample8` demo uses the `XMLTreeViewer` bean. The `XMLSchemaTreeViewer` program lets a user view an `XMLDocument` in a tree format. The user can input an XML-schema document and validate an instance document against the schema. If the document is invalid, invalid nodes are highlighted with an error message.

Also, the program displays a log of all the line information in a separate panel, which enables the user to edit the instance document and revaluated. Test the program with sample files `purchaseorder.xml` and `purchaseorder.xsd`. The instance document `purchaseorder.xml` does not conform to the schema defined in `purchaseorder.xsd`.

You can run the program manually:

```
java XMLSchemaTreeViewer
```

D.2.2.9 Running sample9

The `sample9` demo uses the `SourceViewer` bean. The `XMLSrcViewer` program lets you view an XML document or a DTD, with syntax highlighting. You can validate the document against an input XML schema or DTD. The DTD can be internal or external.

If the validation is successful, then you can view the instance document and XML schema or DTD in the **Source View** pane. If errors were encountered during schema validation, then an error log with line numbers is available in the **Error** pane. The **Source View** pane shows the XML document with error nodes highlighted.You can use sample files `purchaseorder.xml` and `purchaseorder.xsd` for testing XML schema validation with errors. You can use `note_in_dtd.xml` with DTD validation mode to view an internal DTD with validation errors. You can run the program manually:

```
java XMLSrcViewer
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

D.2.2.10 Running sample10

The `sample10` demo shows the use of the `XSDValidator` bean.

The `XSDValidatorSample` program's two input arguments are an XML document and its associated XML schema. The program displays errors occurring during validation, including line numbers.

The following program uses `purchaseorder.xsd` to validate the contents of `purchaseorder.xml`:

```
java XSDValidatorSample purchaseorder.xml purchaseorder.xsd
```

The XML document fails (intentionally) to validate against the schema. The program displays these errors:

```
Sample purchaseorder.xml purchaseorder.xsd
<Line 2, Column 41>: XML-24523: (Error) Invalid value 'abc' for attribute: 'orderDate'
#document->purchaseOrder
<Line 7, Column 27>: XML-24525: (Error) Invalid text 'CA' in element: 'state'
#document->purchaseOrder->shipTo->state->#text
<Line 8, Column 25>: XML-24525: (Error) Invalid text 'sd' in element: 'zip'
#document->purchaseOrder->shipTo->zip->#text
<Line 14, Column 27>: XML-24525: (Error) Invalid text 'PA' in element: 'state'
#document->purchaseOrder->billTo->state->#text
<Line 17, Column 22>: XML-24534: (Error) Element 'coment' not expected.
#document->purchaseOrder->coment
<Line 29, Column 31>: XML-24534: (Error) Element 'shipDae' not expected.
#document->purchaseOrder->items->item->shipDae
```

D.3 Processing XML with XDK JavaBeans

Topics here include processing XML documents asynchronously using the `DOMBuilder` and `XSLTransformer` beans, and comparing XML documents using the `XmlDiff` bean.

D.3.1 Processing XML Asynchronously with the DOMBuilder and XSLTransformer Beans

You can use XDK JavaBeans to perform asynchronous XML processing.

This is explained in XSLTransformer.

The `AsyncTransformSample.java` program shows how to use the `DOMBuilder` and `XSLTransformer` beans. The program implements these methods:

- `runDOMBuilders()`
- `runXSLTransformer()`
- `saveResult()`
- `makeXSLDocument()`
- `createURL()`
- `init()`
- `exitWithError()`

Copyright © 1999, 2025, Oracle and/or its affiliates.

- `asyncTransform()`

The basic architecture of the program is:

1. The program declares and initializes the fields used by the class. The input XSLT stylesheet is hard-coded in the program as `doc.xsl`. The class defines these fields:

```
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
```

2. The `main()` method invokes the `init()` method to perform the initial setup. This method lists the files in the current directory, and if it finds files that end in the extension `.xml`, it adds them to a `Vector` object. The implementation for the `init()` method is:

```
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;
}
```

3. The `main()` method instantiates `AsyncTransformSample`:

```
AsyncTransformSample inst = new AsyncTransformSample();
```

4. The `main()` method invokes the `asyncTransform()` method. The `asyncTransform()` method performs these main tasks:

   a. Invokes `makeXSLDocument()` to parse the input XSLT stylesheet.

   b. Invokes `runDOMBuilders()` to initiate parsing of the instance documents, that is, the documents to be transformed, and then transforms them.

   After initiating the XML processing, the program resumes control and waits while the processing occurs in the background. When the last request completes, the method exits.

   The following code shows the implementation of the `asyncTransform()` method:

```
void asyncTransform () throws Exception
{
   System.err.println (numXMLDocs +
            " XML documents will be transformed" +
            " using XSLT stylesheet specified in " + xslFile +
            " with " +  numXMLDocs + " threads");
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

```
   makeXSLDocument ();
   runDOMBuilders ();

   // wait for the last request to complete
   while (rm.activeFound())
      Thread.sleep(100);
}
```

D.3.1.1 Parsing the Input XSLT Stylesheet

Parsing an XSLT stylesheet is described.

Method `makeXSLDocument()` performs a simple DOM parse of a stylesheet. It does not use asynchronous parsing. The technique is the same as that described in Performing Basic DOM Parsing.

The method follows these steps:

1. Create a new `DOMParser()` object. The following code fragment from `DOMSample.java` shows this technique:

   ```
   DOMParser parser = new DOMParser();
   ```

2. Configure the parser. The following code fragment specifies that white space must be preserved:

   ```
   parser.setPreserveWhitespace(true);
   ```

3. Create a `URL` object from the input stylesheet. The following code fragment invokes the `createURL()` helper method to accomplish this task:

   ```
   xslURL = createURL (xslFile);
   ```

4. Parse the input XSLT stylesheet. The following statement shows this technique:

   ```
   parser.parse (xslURL);
   ```

5. Get a handle to the root of the in-memory DOM tree. You can use the `XMLDocument` object to access every part of the parsed XML document. The following statement shows this technique:

   ```
   xsldoc = parser.getDocument();
   ```

D.3.1.2 Processing the XML Documents Asynchronously

Method `runDOMBuilders()` shows how you can use the `DOMBuilder` and `XSLTransformer` beans to perform asynchronous processing. The parsing and transforming of the XML data occurs in the background.

The method follows these steps:

1. Create a resource manager to manage the input XML documents. The program creates a `for` loop and gets the XML documents. The following code fragment shows this technique:

   ```
   rm = new ResourceManager (numXMLDocs);
   for (int i = 0; i < numXMLDocs; i++)
   {
      rm.getResource();
      ...
   }
   ```

2. Instantiate the DOM builder bean for each input XML document. For example:

Copyright © 1999, 2025, Oracle and/or its affiliates.

```
DOMBuilder builder = new DOMBuilder(i);
```

3. Create a `URL` object from the XML file name. For example:

```
DOMBuilder builder = new DOMBuilder(i);
URL  xmlURL = createURL(basedir + "/" + (String)xmlfiles.elementAt(i));
if (xmlURL == null)
   exitWithError("File " + (String)xmlfiles.elementAt(i) + " not found");
```

4. Configure the DOM builder. The following code fragment specifies the preservation of white space and sets the base URL for the document:

```
builder.setPreserveWhitespace(true);
builder.setBaseURL (createURL(basedir + "/"));
```

5. Add the listener for the DOM builder. The program adds the listener by invoking `addDOMBuilderListener()`.

   The class instantiated to create the listener must implement the `DOMBuilderListener` interface. The program provides a do-nothing implementation for `domBuilderStarted()` and `domBuilderError()`, but must provide a substantive implementation for `domBuilderOver()`, which is the method called when the parse of the XML document completes. The method invokes `runXSLTransformer()`, which is the method that transforms the XML. See Transforming the XML with the XSLTransformer Bean for an explanation of this method.

   The following code fragment shows how to add the listener:

```
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());
      }
   }
);
```

6. Add the error listener for the DOM builder. The program adds the listener by invoking `addDOMBuilderErrorListener()`.

   The class instantiated to create the listener must implement the `DOMBuilderErrorListener` interface. The following code fragment show the implementation:

```
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());
      }
   }
);
```

7. Parse the document. The following statement shows this technique:

Copyright © 1999, 2025, Oracle and/or its affiliates.

```
builder.parse (xmlURL);
System.err.println("Parsing file " + xmlfiles.elementAt(i));
```

D.3.1.2.1 Transforming the XML with the XSLTransformer Bean

When a DOM parse completes, the DOM listener receives notification. Method `domBuilderOver()` implements response behavior for this event. The program passes the DOM to method `runXSLTransformer()`, which initiates XSL transformation.

The method follows these steps:

1. Instantiate the `XSLTransformer` bean. This object performs the XSLT processing. The following statement shows this technique:

   ```
   XSLTransformer processor = new XSLTransformer (id);
   ```

2. Create a new stylesheet object. For example:

   ```
   XSLStylesheet  xsl       = new XSLStylesheet (xsldoc, xslURL);
   ```

3. Configure the XSLT processor. For example, this statement sets the processor to show warnings and configures the error output stream:

   ```
   processor.showWarnings (true);
   processor.setErrorStream (errors);
   ```

4. Add the listener for the XSLT processor. The program adds the listener by invoking `addXSLTransformerListener()`.

   The class instantiated to create the listener must implement the `XSLTransformerListener` interface. The program provides a do-nothing implementation for `xslTransformerStarted()` and `xslTransformerError()`, but must provide a substantive implementation for `xslTransformerOver()`, which is the method called when the parse of the XML document completes. The method invokes `saveResult()`, which prints the transformation result to a file.

   The following code fragment shows how to add the listener:

   ```
   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());
         }
      }
   );
   ```

5. Add the error listener for the XSLT processor. The program adds the listener by invoking `addXSLTransformerErrorListener()`.

Copyright © 1999, 2025, Oracle and/or its affiliates.

The class instantiated to create the listener must implement the `XSLTransformerErrorListener` interface. The following code fragment show the implementation:

```
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());
      }
   }
);
```

6. Transform the XML document with the XSLT stylesheet. The following statement shows this technique:

```
processor.processXSL (xsl, xml);
```

D.3.2 Comparing XML Documents with the XMLDiff JavaBean

You can use XDK JavaBeans to compare the structure and significant content of XML documents.

This is explained in XMLDiff,

The `XMLDiffSample.java` program shows how to use the `XMLDiff` bean. The program implements these methods:

- `showDiffs()`
- `doXSLTransform()`
- `createURL()`

The basic architecture of the program is:

1. The program declares and initializes the fields used by the class. One field is of type `XMLDiffFrame`, which is the class implemented in the `XMLDiffFrame.java` demo. The class defines these fields:

```
protected XMLDocument doc1;   /* DOM tree for first file */
protected XMLDocument doc2;   /* DOM tree for second file */
protected static XMLDiffFrame diffFrame; /* GUI frame */
protected static XMLDiffSample dfxApp;   /* XMLDiff sample application */
protected static XMLDiff xmlDiff;        /* XML diff object */
protected static XMLDocument xslDoc;     /* parsed xsl file */
protected static String outFile = new String("XMLDiffSample.xsl"); /* output
                                                                    xsl file name */
```

2. The `main()` method creates an `XMLDiffSample` object:

```
dfxApp = new XMLDiffSample();
```

3. The `main()` method adds and initializes a `JFrame` to display the output of the comparison. The following code shows this technique:

Copyright © 1999, 2025, Oracle and/or its affiliates.

```
diffFrame = new XMLDiffFrame(dfxApp);
diffFrame.addTransformMenu();
```

4. The `main()` method instantiates the `XMLDiff` bean. The following code shows this technique:

```
xmlDiff = new XMLDiff();
```

5. The `main()` method invokes the `showDiffs()` method. This method performs these tasks:

   a. Invokes `XMLDiff.diff()` to compare the input XML documents.

   b. Generates and displays an XSLT stylsheet that can transform one input document into the other document.

   The following code fragment shows the `showDiffs()` method invocation:

```
if (args.length == 3)
  outFile = args[2];
if(args.length >= 2)
  dfxApp.showDiffs(new File(args[0]), new File(args[1]));
diffFrame.setVisible(true);
```

D.3.2.1 Comparing the XML Files and Generating a Stylesheet

Method `showDiffs()` shows the use of the `XMLDiff` bean.

The method follows these steps:

1. Set the files for the `XMLDiff` processor. The following statement shows this technique:

```
xmlDiff.setFiles(file1, file2);
```

2. Compare the files. The `diff()` method returns a boolean value that indicates whether the input documents have identical structure and content. If they are equivalent, then the method prints a message to the `JFrame` implemented by the `XMLDiffFrame` class. The following code fragment shows this technique:

```
if(!xmlDiff.diff())
{
  JOptionPane.showMessageDialog
  (
    diffFrame,
    "Files are equivalent in XML representation",
    "XMLDiffSample Message",
    JOptionPane.PLAIN_MESSAGE
  );
}
```

3. Generate a DOM for the XSLT stylesheet that shows the differences between the two documents. The following code fragment shows this technique:

```
xslDoc = xmlDiff.generateXSLDoc();
```

4. Display the documents in the `JFrame` implemented by `XMLDiffFrame`. `XMLDiffFrame` instantiates the `XMLSourceView` bean, which is deprecated. The method follows these steps:

   a. Create the source pane for the input documents. Pass the DOM handles of the two documents to the `diffFrame` object to make the source pane:

```
diffFrame.makeSrcPane(xmlDiff.getDocument1(), xmlDiff.getDocument2());
```

   b. Create the pane that shows the differences between the documents. Pass references to the text panes to `diffFrame`:

Copyright © 1999, 2025, Oracle and/or its affiliates.

```
diffFrame.makeDiffSrcPane(new XMLDiffSrcView(xmlDiff.getDiffPane1()),
                          new XMLDiffSrcView(xmlDiff.getDiffPane2()));
```

c. Create the pane for the XSLT stylesheet. Pass the DOM of the stylesheet:

```
diffFrame.makeXslPane(xslDoc, "Diff XSL Script");
diffFrame.makeXslTabbedPane();
```


Copyright © 1999, 2025, Oracle and/or its affiliates.

Glossary

attribute

A property of an element that consists of a name and a value separated by an equal 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 quotation marks. Attributes can reside in the document or document type definition (DTD). Elements may have many attributes but their retrieval order is not defined.

binary XML

An Extensible Markup Language (XML) representation that uses a compact, XML schema-aware format.

callback

A programmatic technique in which one process starts another and then continues. The second process then invokes 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 Procedural Language/Structured Query Language (PL/SQL) that adds the necessary functionality for the database to understand and manipulate a new data type.

Cartridges interface through the Extensibility Framework within the Oracle XML Developer's Kit (XDK) implementation of the Java Architecture for XML Binding (JAXB) specification version 8 or later. Oracle Text is such a cartridge, adding support for reading, writing, and searching text documents stored within the database.

See also Oracle Text.

Cascading Style Sheets (CSS)

See **CSS**.

Copyright © 1999, 2025, Oracle and/or its affiliates.

CDATA

Character data. Text in a document that must not be parsed is included within a `CDATA` section. This allows for the inclusion of characters that would otherwise have special functions, such as `&`, `<`, and `>`. `CDATA` sections can be used in the content of an element or in attributes.

character data (CDATA)

See CDATA.

child element

An element that is wholly contained within another, which is referred to as its parent element. For example `<Parent><Child></Child></Parent>` shows a child element nested within its parent element.

See also parent element.

class generator

A class generator accepts an input file and creates a set of output classes that have corresponding functionality. For the XML class generator, the input file is a DTD or XML schema, and the output is a series of classes that can be used to create conforming XML documents.

CLASSPATH

The operating system environment variable that the Java virtual machine (JVM) uses to find the classes required to run applications.

Common Oracle Runtime Environment (CORE)

See CORE.

CORE

Common Oracle Runtime Environment. The library of functions written in C that enables developers to create code that can be easily ported to virtually any platform and operating system.

CSS

Cascading Style Sheets. A simple mechanism for adding style (fonts, colors, spacing, and so on) to web documents.

data definition language (DDL)

See DDL.

Copyright © 1999, 2025, Oracle and/or its affiliates.

datagram

A text fragment, possibly in XML format, that is returned to the requester embedded in an HTML page from a SQL query processed by the XSQL servlet.

DDL

Data definition language. Statements that define or change a data structure.

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)

See DOM.

document type definition (DTD)

See DTD.

DOM

Document Object Model. An in-memory, tree-based object representation of an XML document that enables programmatic access to its elements and attributes.

The Document Object Model (DOM) object and its interface is a World Wide Web Consortium (W3C) recommendation that specifies the DOM of an XML document, including the application programming interfaces (APIs) for programmatic access. DOM views the parsed document as a tree of objects.

DTD

Document type definition. A set of rules that defines the valid structure of an XML document. DTDs are text files that derive their format from SGML. A DTD can be included in an XML document by using the `DOCTYPE` element or by using an external file through a `DOCTYPE` reference.

See Also

- XML
- SGML
- WML

Copyright © 1999, 2025, Oracle and/or its affiliates.

element

The basic logical unit of an XML document that can serve as a container for other elements, such as children, data, attributes, and their values. Elements are identified by start-tags, such as `<name>`, and end-tags, such as `</name>`, or for empty elements, `<name/>`.

empty element

An element without text content or child elements. It can contain only 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 can represent either another string of characters or special characters that are not part of the document character set. Entities and the text that is substituted for them by the parser are declared in the DTD.

epilog

The closing part of an XML document. The epilog is optional.

Extensible Markup Language (XML)

See XML.

Extensible Stylesheet Language (XSL)

See XSL.

Extensible Stylesheet Language Formatting Objects (XSL-FO)

See XSL-FO.

Extensible Stylesheet Language Transformations (XSLT)

See XSLT.

FOP

Formatting Objects Processor: a print formatter driven by XSL-FO. FOP is a Java application that reads a formatting object tree and renders the resulting pages. The supported output are PDF, PCL, PS, SVG, XML (area tree representation), print, AWT, MIF, and TXT. The primary output target is PDF.

Formatting Objects Processor (FOP)

See FOP.

Copyright © 1999, 2025, Oracle and/or its affiliates.

HTTP

Hypertext Transport Protocol. The set of rules for exchanging files on the World Wide Web. Relative to the TCP/IP suite of protocols, HTTP is an application protocol.

HTTPS

Hypertext Transport Protocol, Secure. The use of Secure Sockets Layer (SSL) as a sublayer under the regular HTTP application layer.

Hypertext Transport Protocol (HTTP)

See HTTP.

Hypertext Transport Protocol, Secure (HTTPS)

See HTTPS.

IDE

Integrated Development Environment. A set of programs designed to aid in the development of software run from a user interface. Oracle JDeveloper is an IDE for Java development that includes an editor, a compiler, a debugger, a syntax checker, and a help system.

infoset

XML Information Set, an abstract data set consisting of several information items. It has at least one information item: the document node, but the infoset is not necessarily valid XML.

The W3C recommendation is at `http://www.w3.org/TR/xml-infoset/`.

instance document

An XML document validated against an XML schema. If the instance document conforms to the rules of the schema, it is said to be valid.

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.

Integrated Development Environment (IDE)

See IDE.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Java EE

Java Platform, Enterprise Edition. The Java platform that defines multitier enterprise computing.

Java

A high-level programming language where applications run in a virtual machine known as a Java Virtual Machine (JVM). The JVM is responsible for all interfaces to the operating system. This architecture lets developers create Java applications that can run on any operating system or platform that has a JVM.

> **See Also**
>
> JVM.

Java Platform, Enterprise Edition (Java EE)

See Java EE.

Java API for XML Processing (JAXP)

See JAXP

Java Architecture for XML Binding (JAXB)

See JAXB.

Java Database Connectivity (JDBC)

See JDBC.

Java Developer's Kit (JDK)

See JDK.

Java Naming and Directory Interface (JNDI)

See JNDI.

Java Specification Request (JSR)

See JSR.

Java Virtual Machine (JVM)

See JVM.

Copyright © 1999, 2025, Oracle and/or its affiliates.

JavaBeans

An independent program module that runs within a Java Virtual Machine (JVM). It is typically used to create user interfaces on a client.

> **See Also**
>
> JVM

JAXB

Java Architecture for XML Binding. An API and tools that map to and from XML documents and Java objects. JAXB is a JSR-31 recommendation.

JAXP

Java API for XML Processing. A programming tool that enables applications to parse and transform XML documents using an API that is independent of a particular XML processor implementation.

JDBC

Java Database Connectivity. The programming API that enables Java applications to access a database through SQL. JDBC drivers are written in Java for platform independence, but are specific to each database.

JDK

Java Developer's Kit. 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.

JNDI

Java Naming and Directory Interface. A programming interface for connecting Java programs to naming and directory services such as DNS, LDAP, and NDS.

JSR

Java Specification Request. A recommendation of the Java Community Process organization (JCP), such as JAXB and XQJ.

Copyright © 1999, 2025, Oracle and/or its affiliates.

JVM

Java Virtual Machine. 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 an intranet, on an application server, or on a database server.

listener

A separate application process that monitors the input process.

marshalling

The process of traversing a Java content tree and writing an XML document that reflects the content of the tree. It is the inverse of unmarshalling.

See also unmarshalling.

node

In XML, the term used to denote each addressable entity in a DOM tree.

notation attribute declaration

In XML, the declaration of a content type that is not part of those understood by the parser. These types include audio, video, and other multimedia.

OASIS

Organization for the Advancement of Structured Information Standards. An organization whose members are chartered with promoting public information standards through conferences, seminars, exhibits, and other educational events. XML and SGML are standards that OASIS is actively promoting.

See Also

- SGML
- XML

Oracle JDeveloper

Oracle JDeveloper is a Java IDE that enables application, applet, and servlet development and includes an editor, compiler, debugger, syntax checker, help system, integrated UML class modeler, and more. It supports XML-based development by including the XDK for Java components in its editor.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Oracle Text

An Oracle tool that provides full-text indexing of documents and the capability to do SQL queries over documents, along with XPath-like searching.

Oracle WebLogic Server

A product 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.

XDK

Oracle XML Developer's Kit. The set of libraries, components, and utilities that provide software developers with the standards-based functionality to XML-enable their applications. In the Java components of XDK, the kit contains an XML parser, an XSLT processor, the XML class generator, the JavaBeans, and the XSQL servlet.

Oracle XML DB

A high-performance XML storage and retrieval technology provided with Oracle Database. It is based on the W3C XML data model.

Oracle XML Developer's Kit (XDK)

See XDK.

ORACLE_HOME

The operating system environment variable that identifies the location for the installation of Oracle components.

Organization for the Advancement of Structured Information Standards (OASIS)

See OASIS.

parent element

An element that surrounds another element, which is referred to as its child element. For example, `<Parent><Child></Child></Parent>` shows a parent element wrapping its child element.

See Also

child element

Copyright © 1999, 2025, Oracle and/or its affiliates.

parsed character data (PCDATA)

See PCDATA.

path name

The name of a resource that reflects its location in the repository hierarchy.

A path name is composed of a root element (the first /), element separators (/), and various subelements (or path elements). A path element can be composed of any character in the database character set except the slash (/) or the backslash (\). These characters have a special meaning for Oracle XML DB. The slash is the default name separator in a path name; the backslash can be used to escape characters.

PCDATA

Parsed character data. The element content consisting of text that must be parsed but is not part of a tag or nonparsed data.

See also tag.

prolog

The opening part of an XML document containing the XML declaration and any DTD or other declarations needed to process the document. The prolog is optional.

repository

The set of database objects, in any schema, that are mapped to path names. There is one root to the repository (/), which contains a set of resources, each with a path name.

> **See Also**
>
> path name

resource

An object in the repository hierarchy.

resource name

The name of a resource within its parent folder. Resource names must be unique (potentially subject to case-insensitivity) within a folder. Resource names are always in the UTF-8 character set (NVARCHAR2).

result set

The output of a SQL query consisting of one or more rows of data.

Copyright © 1999, 2025, Oracle and/or its affiliates.

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 permitted to have only one root element.

> **See Also**
>
> - prolog
> - epilog

SAX

Simple API for XML. 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 refer to an XML document that supports the XML Schema W3C recommendation.

servlet

A Java application that runs in a server, typically a web server or an application server, and performs processing on that server. Servlets are the Java equivalent to CGI scripts.

SGML

Standard Generalized Markup Language. An ISO standard for defining the format of a text document implemented using markup and DTDs.

Simple API for XML (SAX)

See SAX.

Simple Object Access Protocol (SOAP)

See SOAP.

SOAP

Simple Object Access Protocol. An XML-based protocol for exchanging information in a decentralized, distributed environment.

Copyright © 1999, 2025, Oracle and/or its affiliates.

SQL

Structured Query Language. The standard language used to access and process data in a relational database.

SQL/XML

An ANSI specification for representing XML in SQL. Oracle SQL includes SQL/XML functions that query XML: ANSI/ISO/IEC 9075-14:2011, Information technology—Database languages—SQL—Part 14: XML-Related Specifications (SQL/XML).

Standard Generalized Markup Language (SGML)

See SGML.

StAX

Streaming API for XML.

Streaming API for XML (StAX)

See StAX.

Structured Query Language (SQL)

See SQL.

stylesheet

An XML document that consists of XSL processing instructions used by an XSLT processor to transform or format an input XML document into an output XML document.

tag

A single piece of XML markup that delimits the start or end of an element. Tags start with < and end with >. XML includes start-tags (<*name*>), end-tags (</*name*>), and empty tags (<*name*/>), where *name* is the tag name.

TransX Utility

A Java API that simplifies the loading of translated seed data and messages into a database.

Uniform Resource Identifier (URI)

See URI.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Uniform Resource Locator (URL)

See URL.

unmarshalling

The process of reading an XML document and constructing a tree of Java content objects. Each content object corresponds directly to an instance in the input document of the corresponding schema component.

See also marshalling.

URI

Uniform Resource Identifier. The address syntax that is used to create URLs and XPaths.

URL

Uniform Resource Locator. 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 a file name.

valid

The term used to refer to an XML document when its structure and element content is consistent with that declared in its associated DTD or XML schema.

W3C

World Wide Web Consortium. An international industry consortium started in 1994 to develop standards for the World Wide Web. The W3C Web site is located at `http://www.w3c.org`.

See also WWW.

well-formed

An XML document that conforms to the syntax of the XML version declared in its XML declaration. This includes having a single root element and properly nested tags.

Wireless Markup Language (WML)

See WML.

WML

Wireless Markup Language. A tag-based markup language developed for the small display size, reduced memory, and limited processing power of cell phones and other devices that implement the Wireless Application Protocol (WAP) specification. WML documents are XML documents that validate against the WML DTD.

Copyright © 1999, 2025, Oracle and/or its affiliates.

See also DTD.

Working Group (WG)

A W3C committee that is made up of industry members who implement the recommendation process in specific Internet technology areas.

World Wide Web (WWW)

See WWW.

See also W3C.

World Wide Web Consortium (W3C)

See W3C.

See also WWW.

WWW

World Wide Web. A worldwide hypertext system that uses the Internet and the HTTP protocol.

See also W3C.

XDM

The W3C XQuery 1.0 and XPath 2.0 Data Model. A query data model that supports the most XQuery features. The main exceptions are the query prolog, element and attribute constructors, full `FLWOR` syntax, and the typeswitch expression.

XLink

XML Linking Language. It consists of rules that govern the use of hyperlinks in XML documents. The rules are defined by the XML Linking Group, under the W3C recommendation process. This is one of the three languages (XLink, XPointer, and XPath) that XML supports to manage document presentation and hyperlinks.

See Also

- XPath
- XPointer

XML

Extensible Markup Language. An open standard for describing data developed by the World Wide Web Consortium (W3C) using a subset of the SGML syntax and designed for Internet use.

Copyright © 1999, 2025, Oracle and/or its affiliates.

See Also

- SGML
- W3C

XML Base

A W3C recommendation that describes the use of the `xml:base` attribute, which can be inserted in an XML document to specify a base `URI` other than the base URI of the document or external entity. The `URI`s in the document are resolved by the given base.

XML Information Set

See infoset.

XML Linking Language (XLink)

See XLink.

XML Namespaces

Related element names or attributes within an XML document. Namespace syntax and usage are defined by a W3C recommendation. For example, element `<xsl:apply-templates/>` is identified as part of the XSL namespace. Namespaces are declared in an XML document or a DTD before they are used, using this attribute syntax: `xmlns:xsl="http://www.w3.org/TR/WD-xsl"`.

XML parser

A software program that receives an XML document and determines whether it is well-formed and, optionally, valid. The Oracle XML parser supports both SAX and DOM interfaces.

See Also

well-formed

XML Path Language (XPath)

See XPath.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML Pipeline Definition Language

A W3C recommendation that enables you to describe the processing relations between XML resources.

XML Pointer Language (XPointer)

See XPointer.

XML processor

A software program that reads an XML document and processes it, that is, performs actions on the document based on a set of rules. Validity checkers and XML editors are examples of processors.

XML Query (XQuery)

See XQuery.

XML schema

A document written in the XML Schema language.

XML Schema

See XML Schema language.

XML Schema Definition

Equivalent to XML Schema language.

XML Schema language

The XML Schema language, also called XML Schema, is a W3C recommendation for the use of 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.

XML Schema processor

A software program that automatically ensures the validity of XML documents and data used in e-business applications, including online exchanges. It adds simple and complex data types to XML documents, and replaces DTD functionality with an XML schema definition XML document.

XMLSchema-instance namespace

The namespace declaration attribute used to identify an instance document as a member of the class defined by a particular XML schema. You must declare the XMLSchema-instance

Copyright © 1999, 2025, Oracle and/or its affiliates.

namespace by adding a namespace declaration to the root element of the instance document. For example: `xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance`.

XML SQL Utility (XSU)

See XSU.

XMLType

An Oracle data type that stores XML data using object-relational columns or a binary format within a table or view.

XMLType views

A mechanism provided by Oracle XML DB to wrap existing relational and object-relational data in XML format. This is especially useful if, for example, your legacy data is not in XML but you must migrate it to an XML format.

XPath

XML Path Language. The open standard syntax for addressing elements within a document used by XSL and XPointer. XPath is a W3C recommendation. It specifies the data model and grammar for navigating an XML document used by XSLT, XLink, and XML Query.

See Also

- XLink
- XPointer
- XQuery

XPointer

XML Pointer Language. 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.

See Also

- XLink
- XPath

Copyright © 1999, 2025, Oracle and/or its affiliates.

XQJ

XQuery API for Java.

XQSX

XQuery Scripting Extension.

XQuery

XML Query. The ongoing effort of W3C to create a standard for the language and syntax to query XML documents.

XQueryX

XML Syntax for XQuery. XQueryX is an XML representation of an XQuery. See JSR 225.

XQUF

XQuery Update Facility.

XQVM

Oracle XQuery Virtual Machine.

XSL

Extensible Stylesheet Language. The language used within stylesheets to transform or render XML documents. Two W3C recommendations cover XSL stylesheets: XSL Transformations (XSLT) and XSL Formatting Objects (XSL-FO).

- XSLT is a language for transforming one XML document into another.
- XSL-FO is an XML vocabulary for specifying the presentation of an XML document.

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.

See Also

- XSLT
- XSL-FO

XSL-FO

XSL Formatting Objects. Also known as Extensible Stylesheet Language Formatting Objects, and XSLFO. The W3C standard specification that defines an XML vocabulary for specifying formatting semantics.

Copyright © 1999, 2025, Oracle and/or its affiliates.

> **See Also**
>
> FOP.

XSL Formatting Objects (XSL-FO)

See XSL-FO.

XSL Transformations (XSLT)

See XSLT.

XSLT

Extensible Stylesheet Language Transformations. Also known as XSL-T. The XSL W3C standard specification that defines a transformation language to convert one XML document into another.

XSLT Virtual Machine (XVM)

Also XSLT VM. See XVM.

XSLTVM

Also XSLT VM. See XVM.

XSQL pages

XML pages that contain instructions for the XSQL servlet.

XSQL pages publishing framework

See XSQL servlet.

XSQL servlet

A Java-based servlet that can dynamically generate XML documents from one or more SQL queries and optionally transform the documents in the server with an XSLT stylesheet.

XSU

XML SQL Utility. An Oracle utility that can generate an XML document (string or DOM) when given a SQL query or a JDBC `ResultSet` object. XSU can also extract the data from an XML document, and then insert, update, or delete rows in a database table.

Copyright © 1999, 2025, Oracle and/or its affiliates.

XVM

XSLT Virtual Machine. Also known as XSLTVM and XSLT VM. The Oracle XSLT Virtual Machine is the software implementation of a CPU designed to run compiled XSLT code. The virtual machine concept assumes a compiler compiling XSLT stylesheets to a program of bytecodes, or machine instructions for the XSLT CPU.

Copyright © 1999, 2025, Oracle and/or its affiliates.

Index

Symbols

.NET, *16*

B

binary XML
    C, *1*
    decoding, *5*
    encoding, *4*
    Java, *1*
    models for using, *2*
    saving text as, *21*
    storage format, *1*
    terminology, *2*
    using Java, *6*
    vocabulary management, *5*
binary XML decoder, *8*
binary XML encoder, *7*
Built-in Action Handler, *21*
Built-in Action Handler, XSQL, *21*

C

C API, *11*
C compile-time environment on UNIX
    setting up, *5*
C components
    demos, *1*, *6*, *8*, *4*
    directory structure, *1*
    globalization support, *12*
    installation, *1*
    runtime environment on Windows, *7*
    samples, *1*, *6*, *8*, *4*
    setting up Windows environment, *6*
    setting up Windows environment variables, *7*
    with Visual C/C++ on Windows, *9*
C environment variables on UNIX, *4*
C libraries
    contents, *3*
C runtime environment on UNIX, *4*
C++ class generator, *5*
C++ interface, *1*
Class Generator
    XML C++, *1*
CLASSPATH
    XSQL Pages, *6*
CLASSPATH environment variable
    for XQJ, *1*
command-line interface
    oraxml, *14*, *8*
Connection Definitions, *7*
custom connection manager, *25*
custom entity resolver
    example, *2*

D

Data Provider for .NET, *16*
data variables into XML, *48*
DB Access JavaBean, *D-3*
decoding binary XML, *5*
Default SQL to XML Mapping, *27*
demos
    C components, *1*, *6*, *8*, *4*
directory structure
    C, *1*
document creation Java APIs, *3*
DOM
    creating in Java, *1*
    specifications, *2*
DOMBuilder Bean, *D-2*
DTDs
    external, *49*

E

encoding binary XML, *4*
entity resolver
    other entity types, *12*
entity resolver framework, *2*
error messages
    DLF, *B-1*
    DML, *C-2*
    DOM, *A-10*
    generic, *C-1*
    JAXB, *A-49*
    query, *C-2*
    schema component constraint, *A-37*
    schema representation constraint, *A-32*
    TransX, *B-3*

Copyright © 1999, 2025, Oracle and/or its affiliates.

error messages *(continued)*
XML parser, *A-1*
XML pipeline, *A-47*
XML schema validation, *A-22*
XPath, *A-17*
XSL transformation, *A-15*
XSQL server pages, *A-47*
examples of document creation in Java, *3*
external storage
example, *16*

F

FileReader not for system files, *52*
FOP
serializer, *8*
serializer to produce PDF, *16*

G

generated XML
customizing, *30*
generating XML, *12*
using XSU command line, getXML, *12*
getXML, *12*
globalization support
for the C components, *12*

H

HTML Form Parameters, *26*
HTTP Parameters, *25*
HTTP POST method, *30*

I

informational messages
TransX, *B-3*
insert, XSU, *32*
installation
C components, *1*
invalid characters, *55*

J

JAR files, DTDs, *49*
Java classes deprecated, *2*
Java components
creating a DOM, *1*
environment in Windows, *7*
installation, *1*
parsing, *1*
Java diff operations, *3*
append-node, *4*
delete-node, *6*
Java diff operations *(continued)*
examples, *6*
insert-node-before, *5*
Java diff output schema
xdiff.xsd, *10*
Java Specification Request
225, *1*, *6*
Java XML diffing library, *1*
JAXB
class generator, *5*
compared with JAXP, *1*, *4*
features not supported, *9*
marshalling and unmarshalling, *1*
validating, *1*
what is, *4*
JAXP
compared with JAXB, *1*
JAXP (Java API for XML Processing), *40*
JDBC driver, *2*
JSR-225, *1*, *6*

M

make.bat file
editing on Window for C environment, *8*
mapping
primer, XSU, *27*
messages
assertion, *B-4*
Microsoft .NET, *16*

N

no rows exception, *26*

O

OCI and the XDK for C, *21*
OCI examples, *23*
Oracle JDeveloper, *14*
Oracle JVM, *11*
Oracle XML Developer’s Kit components, *1*
Oracle XML Developer’s Kit version
using C, *6*
using C++, *3*
using Java, *9*
oracle.xml.diff package, *1*
OracleXml namespace, *1*
orastream functions, *13*
oraxml, *14*, *8*
oraxsl
command-line interfaces, *6*
Out Variable, using xsql
dml, *27*

Copyright © 1999, 2025, Oracle and/or its affiliates.

P

Package Classes, *2*
Parser for Java, *1*
  constructor extension functions, *12*
  oraxsl, *6*
  return value extension function, *12*
  static and nonstatic methods, *11*
  supported database, *11*
  using DTDs, *49*
Parser for Java, overview, *11*
PDF results using FOP, *8*
Pipeline Definition Language, *1*

S

samples
  C components, *1*, *6*, *8*, *4*
SAX, *1*
security, XSQL Pages, *30*
select
  with XSU, *32*
servlet, XSQL, *1*, *1*
SOAP
  C clients, *3*
  C examples, *6*
  C Functions, *4*
  for C, *1*
  server, *4*
  what is, *1*
static context
  default initial values, *29*
streaming evaluation, *15*
  example, *15*
streaming validator, *5*
  opaque mode, *7*
  transparent mode, *5*
string data, *55*

T

text node normalization, *2*
TransX Utility, *1*

U

Unicode in a system file, *52*
unified C API for XDK and Oracle XML DB, *11*
unified Java API, *1*
Unified Java API, *1*
Unified Java API new objects and methods, *3*
UNIX environment for C components
  configuring, *3*
update, XSU, *33*
updating queries, *18*
updating query
  example, *18*
UTF-16 Encoding, *54*
UTF-8 output, *53*

V

validation
  auto validation mode, *8*
  DTD validating Mode, *8*
  partial validation mode, *8*
  schema validation, *8*
  schema validation mode, *8*
Visual C/C++, *9*
Visual Studio, *9*

W

W3C DOM, *1*
Windows, *6*
  C components
    with Visual C/C++, *9*
  C libraries, *6*
  editing make.bat file, *8*
  setting up C environment variables, *7*
Windows environment for C components
  setting up, *6*
WML Document, *25*

X

Xdiff instance document, *4*
Xdiff schema, *6*
xdiff.xsd, *10*
XDK
  JAR files for XQJ, *1*
XDK components, *1*
XDK version
  using C, *6*
  using C++, *3*
  using Java, *9*
XML Base, *1*
XML C++ Class Generator, *1*
XML DB
  JAR files for XQJ, *1*
XML diffing methods
  Java, *1*
XML documents
  generating from C, *12*
  generating from C++, *13*
  generating from Java, *10*
XML equal methods
  Java, *1*, *10*
XML hash methods
  Java, *1*, *10*

Copyright © 1999, 2025, Oracle and/or its affiliates.

XML input documents
comparing and contrasting, 3
XML namespace prefixes
ignoring differences, 2
XML Namespaces 1.0, 1
XML output in UTF-8, 53
XML parser
oraxml command-line interface, 14, 8
XML parser for C
sample programs, 6, 8
XML pull parser
example, 19
XML Pull Parser error handling, 18
XML Pull Parser for C, 16
XML Schema
explained, 3
processor for Java
how to run the sample program, 4, 9, 6, 7, 5
XML schema for C
sample programs, 4
XML SQL Utility (XSU), 7
connecting with OCI* JDBC driver, 2
customizing generated XML, 30
dependencies and installation, 2
explained, 2
getXML command line, 12
inserts, 32
mapping primer, 27
selects, 32
updates, 33
XML Syntax for XQuery (XQueryX), 28
xmlcg usage, 1
XMLCompress JavaBean, D-4
XMLDBAccess JavaBean, D-3
XmlDiff
command-line options for C, 2
XMLDiff
example in C, 9
XMLDiff in C, 1
XMLDiff JavaBean, D-4
XmlHash
example in C, 13
XMLNode.selectNodes() method, 47
XmlPatch
command-line options for C, 11
XQJ, 28, 1
entity resolver framework, 2
updating queries, 18
XQuery API for Java, 1
XQuery Update Facility, 18
XQJ implementation-defined items
support in XDK, 29
XQuery API for Java, 1
XQuery API for Java (XQJ), 28, 1
XQuery implementation-defined items
support in XDK, 29
XQuery language, 28
XQuery optional features
support in XDK, 28
XQuery processor for Java, 1
standards and specifications, 28
streaming evaluation, 15
using external storage, 16
XQuery Update Facility, 18, 28
XQuery Update Facility implementation-defined items
support in XDK, 29
XQueryX, 28
XSL Transformation (XSLT) Processor, 4
XSL Transformation (XSLT) Processor for Java, 3, 3, 3
XSL Transformations Specifications, 3
XSLT
XSLTransformer bean, D-7
XSLT compiler, 1
XSLT processor, 3
XSLT Processor for Java
hints for using, 14
XSLTransformer JavaBean, D-2
XSLValidator JavaBean, D-5
XSQL
action handler errors, 11
advanced topics, 1
built-in action handler elements, 21
connection, 28, 29
current page name, 29
errors, 29
setting up demos, 10
SOAP support, 28
stylesheets, 2
two queries, 26
XSQL action elements
((lt))xsql((colon))action((gt)), 6
((lt))xsql((colon))delete-request((gt)), 8
((lt))xsql((colon))dml((gt)), 9
((lt))xsql((colon))if-param((gt)), 10
((lt))xsql((colon))include-owa((gt)), 12
((lt))xsql((colon))include-param((gt)), 13
((lt))xsql((colon))include-posted-xml((gt)), 14
((lt))xsql((colon))include-request-params((gt)), 14
((lt))xsql((colon))include-xml((gt)), 16
((lt))xsql((colon))include-xsql((gt)), 17
((lt))xsql((colon))insert-param((gt)), 19
((lt))xsql((colon))insert-request((gt)), 20
((lt))xsql((colon))query((gt)), 21
((lt))xsql((colon))ref-cursor-function((gt)), 24
((lt))xsql((colon))set-cookie((gt)), 26
((lt))xsql((colon))set-page-param((gt)), 28
((lt))xsql((colon))set-session-param((gt)), 30

Copyright © 1999, 2025, Oracle and/or its affiliates.

XSQL action elements *(continued)*
((lt))xsql((colon))set-stylesheet-param((gt)), *32*
((lt))xsql((colon))update-request((gt)), *33*
XSQL Pages security, *30*
XSQL servlet
hints, *25*
XSQL Servlet examples, *8*
XSU
generating XML, *12*
mapping primer, *27*
usage guidelines, *27*
XSU (XML SQL Utility), *7*
XSU usage techniques, *27*
XVM
XSLT compiler, *3*
XVM (XSLT Virtual Machine) processor, *1*

Copyright © 1999, 2025, Oracle and/or its affiliates.