Oracle9*i*

XML Developer's Kits Guide - XDK

Release 2 (9.2)

March 2002

Part No. A96621-01

ORACLE®

Oracle9*i* XML Developer's Kits Guide - XDK, Release 2 (9.2)

Part No. A96621-01

Primary Author: Jack Melnick

Contributing Authors: Mark Bauer, Shelley Higgins, Steve Muench, Mark Scardina, Jinyu Wang

Contributors: Sandeepan Banerjee, Kishore Bhamidipati, Bill Han, K. Karun, Murali Krishnaprasad, Bruce Lowenthal, Anjana Manian, Meghna Mehta, Nick Montoya, Ravi Murthy, Den Raphaely, Blaise Ribet, Tarvinder Singh, Tomas Saulys, Tim Yu, Jim Warner, Simon Wong, Kongyi Zhou

Graphic Designer: Valarie Moore

Contents

2 Getting Started with XDK for Java and JavaBeans

3 Getting Started with XDKs for C/C++ and PL/SQL

5 XSLT Processor for Java

6 XML Schema Processor for Java

7 XML Class Generator for Java

8 XML SQL Utility (XSU)

10 XDK JavaBeans

11 Using XDK and SOAP

14 XSLT Processor for C

15 XML Schema Processor for C

16 XML Parser for C++

17 XSLT Processor for C++

18 XML Schema Processor for C++

19 XML Class Generator for C++

Part IV XDK for PL/SQL

20 XML Parser for PL/SQL

25 Introduction to BC4J

26 Introduction to UIX

A XDK for Java: Specifications and Quick References

B XDK for PL/SQL: Specifications

Send Us Your Comments

Oracle9*i* XML Developer's Kits Guide - XDK, Release 2 (9.2)

Part No. A96621-01

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

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

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

- Electronic mail: infodev_us@oracle.com
- FAX: (650) 506-7227 Attn: Server Technologies Documentation Manager
- Postal service:
  Oracle Corporation
  Server Technologies Documentation
  500 Oracle Parkway, Mailstop 4op11
  Redwood Shores, CA 94065
  USA

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

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

Preface

The Preface has the following sections:

About this Guide

This manual describes Oracle9*i*'s XML-enabled database technology. It describes how XML data can be stored, managed, and queried in the database using Oracle XML-enabled technology and the appropriate Oracle development tools.

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

Other Documentation on XML

For more about building XML applications:

See Also:

- *Oracle9i XML Database Developer's Guide - Oracle XML DB*
- *Oracle9i XML API Reference - XDK and Oracle XML DB*
- *Oracle9i Application Developer's Guide - Advanced Queuing*

Examples and Sample Code

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

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

How to Order this Manual

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

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

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

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

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

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

`http://otn.oracle.com/admin/account/membership.html`

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

`http://otn.oracle.com/docs/index.htm`

To access the database documentation search engine directly, please visit

`http://tahiti.oracle.com`

Downloading Release Notes, Installation Guides, White Papers

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

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

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

`http://otn.oracle.com/docs/index.htm`

To access the database documentation search engine directly, please visit

`http://tahiti.oracle.com`

How to Access this Manual On-Line

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

- On the Document CD that accompanies your Oracle9i software CD
- From Oracle Technology Network (OTN) at `http://otn.oracle.com/docs/index.html`, under Data Server (or whatever other product you have). For example, select Oracle9i > General Documentation Release 1 (9.0.1) (or whatever other section you need to

specify). Select HTML then select HTML or PDF for your particular of interest, such as, “Oracle Documentation Library”. Note that you may only be able to locate the prior release manuals at this site.

Conventions

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

- Conventions in Text
- Conventions in Syntax and Code Examples

Conventions in Text

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

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

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

Conventions in Syntax and Code Examples

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

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

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

Convention	Meaning	Example
[ ]	In syntax examples, brackets enclose one or more optional items. Do not enter the brackets.	`DECIMAL (`*digits* `[ , `*precision* `])`
{ }	In syntax examples, braces enclose two or more items, one of which is required. Do not enter the braces.	`{ENABLE | DISABLE}`

Convention	Meaning	Example
|	In syntax examples, a vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	{ENABLE | DISABLE} [COMPRESS | NOCOMPRESS]
. . .	Horizontal ellipsis points indicate either: ■ That we have omitted parts of the code that are not directly related to the example ■ Or, in syntax examples, that you can enter more arguments	CREATE TABLE ... AS *subquery*; SELECT *col1*, *col2*, ... , *coln* FROM employees;
. . .	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	SQL> SELECT NAME FROM V$DATAFILE; NAME ---------- /fsl/dbs/tbs_01.dbf /fs1/dbs/tbs_02.dbf . . . /fsl/dbs/tbs_09.dbf 9 rows selected.
Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown.	acctbal NUMBER(11,2); acct CONSTANT NUMBER(4) := 3;
Italics	Italicized text indicates placeholders or variables for which you must supply particular values.	CONNECT SYSTEM/*system_password* DB_NAME = *database_name*
UPPERCASE	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase.	SELECT last_name, employee_id FROM employees; SELECT * FROM USER_TABLES; DROP TABLE hr.employees;

Convention	Meaning	Example

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

Documentation Accessibility

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

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

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

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

What's New in XDK?

These sections describe the new features in the following releases:

- XDK Features Introduced with Oracle9i, Release 2 (9.2)
- XDK Features Introduced with Oracle9i, Release 1 (9.0.1)
- XDK Features Introduced with Oracle8i Release 3 (8.1.7)

XDK Features Introduced with Oracle9*i*, Release 2 (9.2)

XML Schema Processor for Java

Supports the W3C Schema recommendation.

Schema Identity-constraint validation no longer needs external DocumentBuilder.

XSL Stylesheets

Support for threadsafe XSLStylesheet objects.

XSQL Servlet

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

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

Simplified inclusion of XML from CLOB and VARCHAR2 Columns.

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

Support for the Apache FOP 0.19 release.

Supports Immediately Read Values Set as Cookies.

Supports Setting Multiple Parameter Values with a Single SQL Statement.

Class Generator for Java

Data Binding Feature is added in this release to the DTD Class Generator.

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

XDK for Java

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

DOM level compression support.

Oracle SOAP APIs added.

SAX2 Extension support in the Java XML Parser.

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

Oracle TransX Utility aids loading data and text.

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

XML Compression now supported in the Java XML Parser.

XDK for C

Released on Linux.

XDK for C++

Released on Linux.

XDK for JavaBeans

New XMLDiff Bean.

Internal DTD support is added to the SourceViewer Bean.

OTN

New XDK Live demo is online at:

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

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

New XDK technical Paper for "Building Server-Side XML Schema Validation" is online at:

`http://otn.oracle.com/tech/xml/xdk_sample/xdksample_093001i.html`

XDK Features Introduced with Oracle9*i*, Release 1 (9.0.1)

Here are the new XDK features in Oracle9*i* Release 1 (9.0.1):

XDK for Java

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

  See:

  - Chapter 4, "XML Parser for Java"
  - Chapter 6, "XML Schema Processor for Java"

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

See: Chapter 7, "XML Class Generator for Java"

- **XSQL Servlet and Pages**
  - Support for Database Bind Variables. Now both lexical subsitution and true database bind variables are supported for improved performance.
  - Support for PDF Output Using Apache FOP. You can now combine XSQL Pages with the Apache FOP processor to produce Adobe PDF output from any XML content.
  - Trusted Host Support for XSLT Stylesheets. New security features insure that stylesheets cannot be executed from non-trusted hosts.
  - Full Support for Non-Oracle JDBC Drivers. Now all query, insert, update, and delete features with with both Oracle and Non-Oracle JDBC drivers.
  - Process Dynamically Constructed XSQL Pages. The XSQLRequest API can now process programmatically constructed XSQL pages.
  - Use a Custom Connection Manager. You can now implement your own Connection Manager to handle database connections in any way you like.
  - Produce Inline XML Schema. You can now optionally produce an inline XML Schema that describes the structure of your XML query results.
  - Set Default Date Format for Queries. You can now supply a date format mask to change the default way date data is formatted.
  - Write Custom Serializers. You can create and use custom serializers that control what and how the XSQL page processor will return to the client.
  - Dynamic Stylesheet Assignment. Assign stylesheets dynamically based on parameters or the result of a SQL query.
  - Update or Delete Posted XML. In addition to inserting XML, now updating and deleting is also supported.
  - Insert or Update Only Targeted Columns. You can now explicitly list what columns should be included in any insert or update request.
  - Page-Request Scoped Objects. Your action handlers can now get/set objects in the page request context to share state between actions within a page.
  - Access to ServletContext. In addition to accessing the HttpRequest and HttpResponse objects, you can also access the ServletContext.

    See: Chapter 9, "XSQL Pages Publishing Framework"

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

    See: Chapter 10, "XDK JavaBeans"

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

    See: Chapter 15, "XML Schema Processor for C"

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

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

- **XDK for PL/SQL**
  - Improved XSLT performance

    See: Chapter 20, "XML Parser for PL/SQL"

XML SQL Utility (XSU) Features

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

columns should be mapped to an XML attribute instead of an XML element.

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

See: Chapter 8, "XML SQL Utility (XSU)"

XDK Features Introduced with Oracle8*i* Release 3 (8.1.7)

New XDK features introduced in Oracle8i, Release 3 (8.1.7) were enhanced and improved versions of the following components:

- **XDK for Java**
- **XDK for C**
- **XDK for C++**
- **XDK for PL/SQL**
- **XML SQL Utility**

Part I

XML Developer's Kits (XDK)

Part I of the book introduces you to Oracle XML-enabled technology and features, Oracle XML Developer's Kits (XDKs) and XML components, and how to install the XDKs. Part I contains the following chapters:

1

Overview of XML Developer's Kits and Components

This chapter contains the following sections:

Oracle XML Components: Overview

Oracle9*i* provides several components, utilities, and interfaces you can use to take advantage of XML technology in building your Web-based database applications. Which components you use depends on your application requirements, programming preferences, development, and deployment environments.

Starting with XDK 9.0.2 (shipped with iAS v2) and XDK 9.2 (shipped with Oracle9*i* Release 2), XSLStylesheet is thread-safe and can be used across threads in multiple `XSLProcessor.processXSL` calls. But XSLProcessor, a light-weight object, will not be made thread safe.

The following XML components are provided with Oracle9*i* and Oracle9*i* Application Server:

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

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

- Figure 1–8, "Generating XML Documents Using XDK for Java"
- Figure 1–9, "Generating XML Documents Using XDK for C"
- Figure 1–10, "Generating XML Documents Using XDK for C++"
- Figure 1–11, "Generating XML Documents Using XDK for PL/SQL"

Table 1–1 XDK Component Descriptions

XDK Component	Languages	Description
XML Parser	Java, C, C++, PL/SQL	Creates and parses XML using Internet standard DOM and SAX interfaces.
XSLT Processor	Java, C, C++, PL/SQL	Transforms or renders XML into other text-based formats such as HTML and WML
XML Schema Processor	Java, C, C++, PL/SQL	Enables the use of XML simple and complex datatypes by means of your XML Schema definitions.

Table 1–1 XDK Component Descriptions (Cont.)

XDK Component	Languages	Description
XML Class Generator	Java, C++	Automatically generates Java and C++ classes from DTDs and XML Schemas to send XML data from Web forms or applications.
XML Transviewer JavaBeans	Java	View and transform XML documents and data through Java components.
XML SQL Utility (XSU)	Java, PL/SQL	Generates XML documents, DTDs, and XML Schemas from SQL queries.
XSQL Servlet	Java	Combines XML, SQL, and XSLT in the server to deliver dynamic Web content.
TransX Utility	Java	Loads data encapsulated in XML into the database with additional SQL functionality useful for installations.
Oracle SOAP Server	Java	See also Chapter 11, "Using XDK and SOAP"
XML Compressor	Java	See also "XML Compressor" on page 4-10.

Development Tools and Other XML-Enabled Oracle9*i* Features

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

Oracle Text: A querying, search and retrieval tool.

Oracle JDeveloper9i and BC4J: JDeveloper9*i* is an integrated development tool for building Java web-based applications. Oracle Business Components for Java (BC4J) is a Java, XML-powered framework that enables productive development, portable deployment, and flexible customizing of multitier, database-savvy applications from reusable business components. These applications can be deployed as CORBA Server Objects or EJB Session Beans on enterprise-scale server platforms supporting Java technology.

See Also:

- Chapter 21, "XSLT Processor for PL/SQL"
- Chapter 22, "XML Schema Processor for PL/SQL"

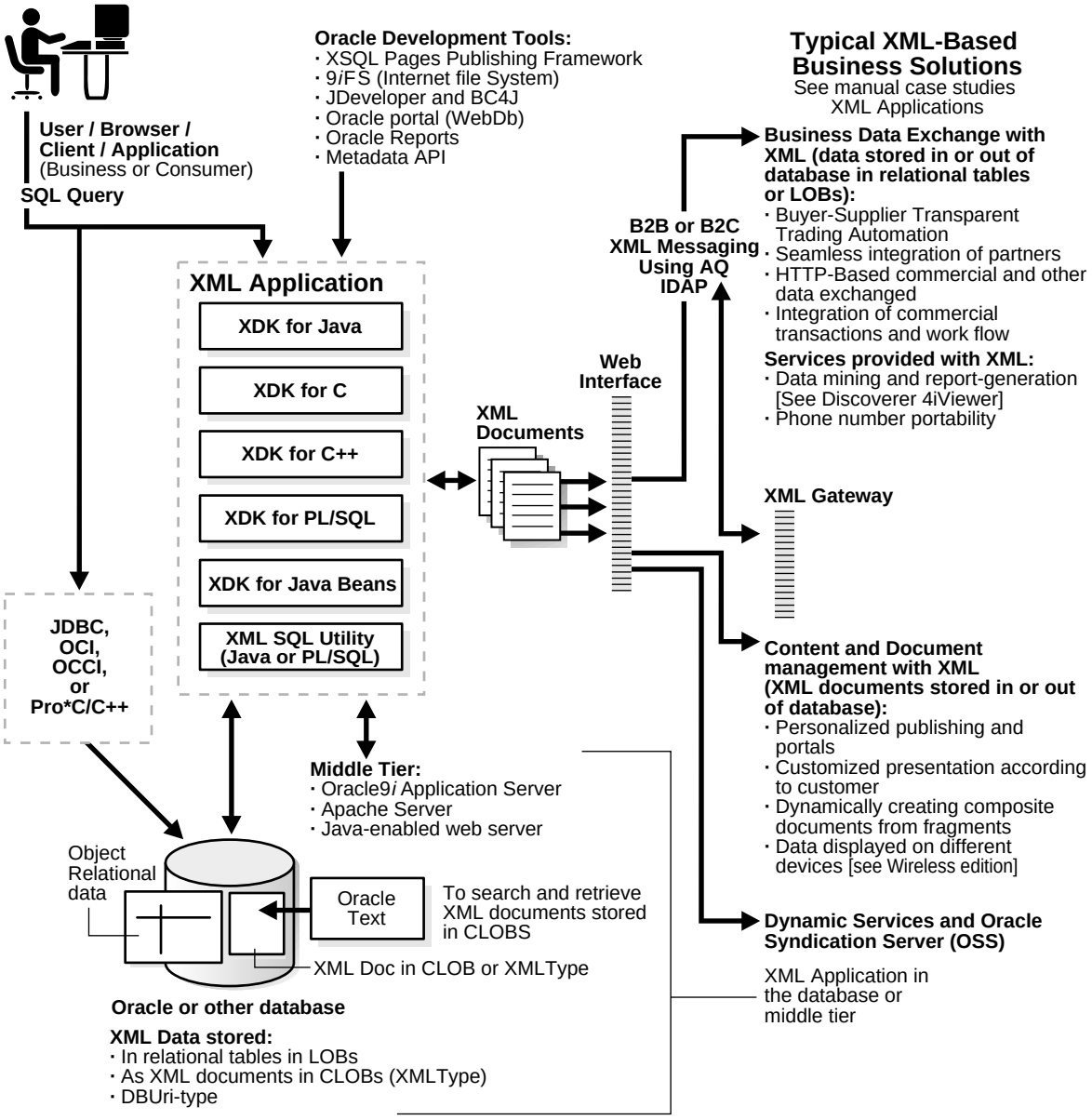
- *Oracle9i Internet File System (9iFS)*: An application interface in which data can be viewed as documents and the documents can be treated as data. *9iFS* is a simple way for developers to work with XML, where *9iFS* serves as the repository for XML. *9iFS* can perform the following tasks on XML documents:

- Automatically parse XML and store content in tables and columns
- Render the XML file's content

  See Also: *Oracle9i XML Case Studies and Applications*, the chapter, "Using Internet File System (9*i*FS) to Build XML Applications".

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

Figure 1–1 Oracle XML Components and E-Business Solutions: What Is Involved



XDK for Java

XDK for Java is composed of the following components:

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

XDK for JavaBeans

XDK for JavaBeans is composed of the following component:

- **XML Transviewer JavaBeans.** View and transform XML documents and data through Java
- **XMLDiff Bean.** The XML Diff Bean performs a tree comparison on two XML DOM trees. It displays the two XML trees and shows the differences between the XML trees.

XDK for C

XDK for C is composed of the following component:

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

XDK for C++

XDK for C++ is composed of the following:

- **XML Parser for C++.** Creates and parses XML using Internet standard DOM and SAX interfaces. Includes an **XSL Transformation (XSLT) Processor** that transforms XML to XML or other text-based formats, such as HTML.
- **XML Schema Processor for C++.** A companion component to XML Parser for C++. It enables support for simple and complex datatypes in XML applications with **Oracle9***i*. The Schema Processor supports the XML Schema Working Draft.
- **XML C++ Class Generator:** Creates source files from an XML DTD or XML Schema definition.
- **XSLT Processor.**Transforms or renders XML into other text-based formats such as HTML and WML.

XDK for PL/SQL

XDK for PL/SQL is composed of the following:

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

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

- **XSLT Processor.**
- **XML Schema Processor.** Transforms or renders XML into other text-based formats such as HTML and WML**.**

XML Parsers

The Oracle XML parser includes implementations in C, C++, PL/SQL, and Java for the full range of platforms on which Oracle9*i* runs.

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

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

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

The parsers are available on all Oracle platforms.

Figure 1–2 illustrates the Oracle XML Parser for Java. Figure 1–3 illustrates the Oracle XML parsers' overall functionality.

> **See Also:** Chapter 4, "XML Parser for Java" and Chapter A, "XDK for Java: Specifications and Quick References".

Figure 1–2 Oracle XML Parser for Java

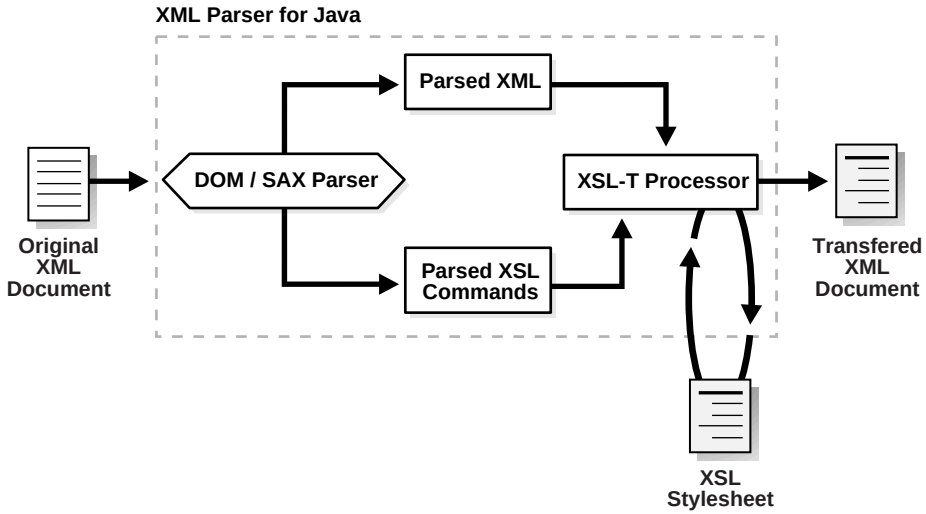
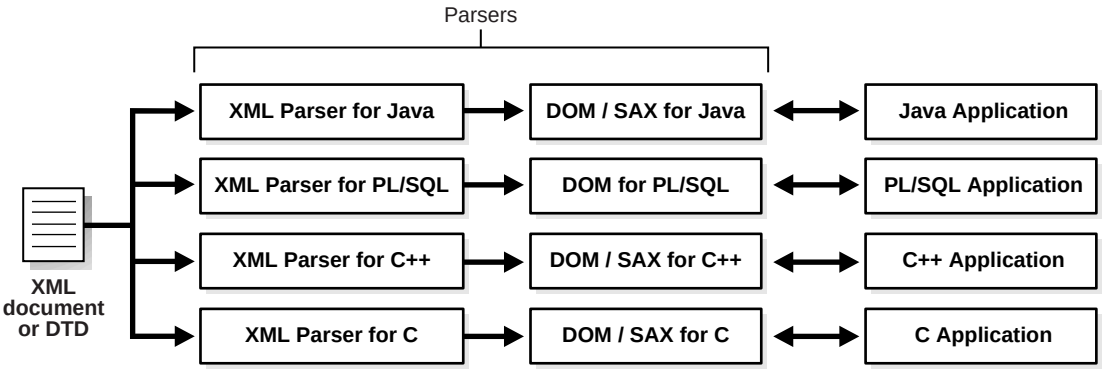


Figure 1–3 The XML Parsers: Java, C, C++, PL/SQL



XSL Transformation (XSLT) Processor

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

- Enables standards-based transformation of XML information inside and outside the database on any platform.
- Supports Java extensibility and for additional performance comes natively compiled from Oracle8*i* Release 3 (8.1.7) and higher.

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

How to use the XSLT Processor is described in Chapter 4, "XML Parser for Java".

> **See Also:** Chapter A, "XDK for Java: Specifications and Quick References".

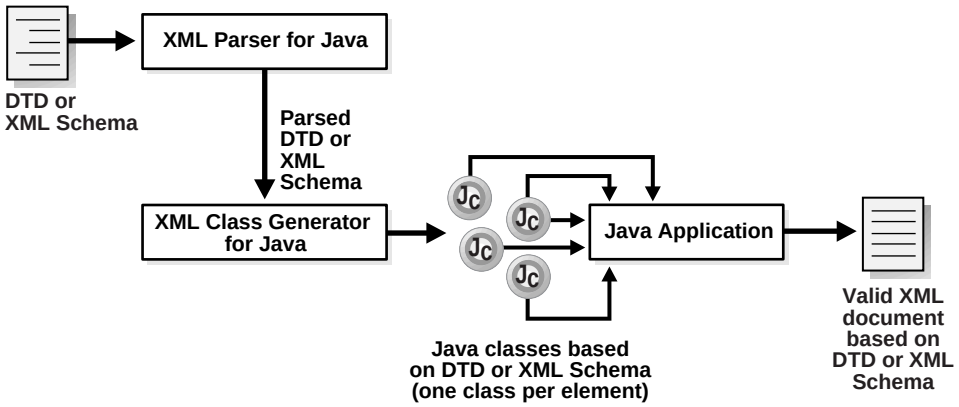
XML Class Generator

XML Class Generator creates a set of Java or C++ classes for creation of XML documents corresponding to an input DTD or XML Schema. Figure 1–4 shows Oracle XML Class Generator functionality.

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

- Chapter 7, "XML Class Generator for Java"
- Chapter 19, "XML Class Generator for C++"

Figure 1–4 Oracle XML Java Class Generator



XML Transviewer JavaBeans

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

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

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

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

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

As standard JavaBeans, they can be used in any graphical Java development environment, such as Oracle JDeveloper. The Oracle XML Transviewer Beans functionality is described in Chapter 10, "XDK JavaBeans".

Oracle XSQL Page Processor and Servlet

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

You can use XSQL Servlet to perform the following tasks:

- Build dynamic XML data pages from the results of one or more SQL queries and serve the results over the Web as XML datagrams or HTML pages using server-side XSLT transformations.

- Receive XML posted to your web server and insert it into your database.

Servlet Engines That Support XSQL Servlet

XSQL Servlet has been tested with the following servlet engines:

- Allaire JRun 2.3.3
- Apache 1.3.9 with JServ 1.0 and 1.1
- Apache 1.3.9 with Tomcat 3.1 Beta1 Servlet Engine
- Apache Tomcat 3.1 Beta1 Web Server + Servlet Engine
- Caucho Resin 1.1
- NewAtlanta ServletExec 2.2 for IIS / PWS 4.0
- Oracle9*i* Lite Web-to-Go Server
- Oracle Application Server 4.0.8.1 (with JSP Patch)
- Oracle8*i* 8.1.7 Beta Aurora and Oracle9*i* Servlet Engine and higher
- Sun JavaServer Web Development Kit (JSWDK) 1.0.1 Web Server

JavaServer Pages Platforms That Support XSQL Servlet

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

- Apache 1.3.9 with Tomcat 3.1 Beta1 Servlet Engine
- Apache Tomcat 3.1 Beta1 Web Server + Tomcat 3.1 Beta1 Servlet Engine
- Caucho Resin 1.1 (Built-in JSP 1.0 Support)
- NewAtlanta ServletExec 2.2 for IIS / PWS 4.0 (Built-in JSP 1.0 Support)
- Oracle9*i* Lite Web-to-Go Server with Oracle JSP 1.0
- Oracle8*i* 8.1.7 Beta Aurora and Oracle9*i* Servlet Engine with Oracle JSP 1.0 and higher
- Any Servlet Engine with Servlet API 2.1+ and Oracle JSP 1.0

In general, it should work with the following:

- Any servlet engine supporting the Servlet 2.1 specification or higher

- Oracle JSP 1.0 reference implementation or functional equivalent from another vendor

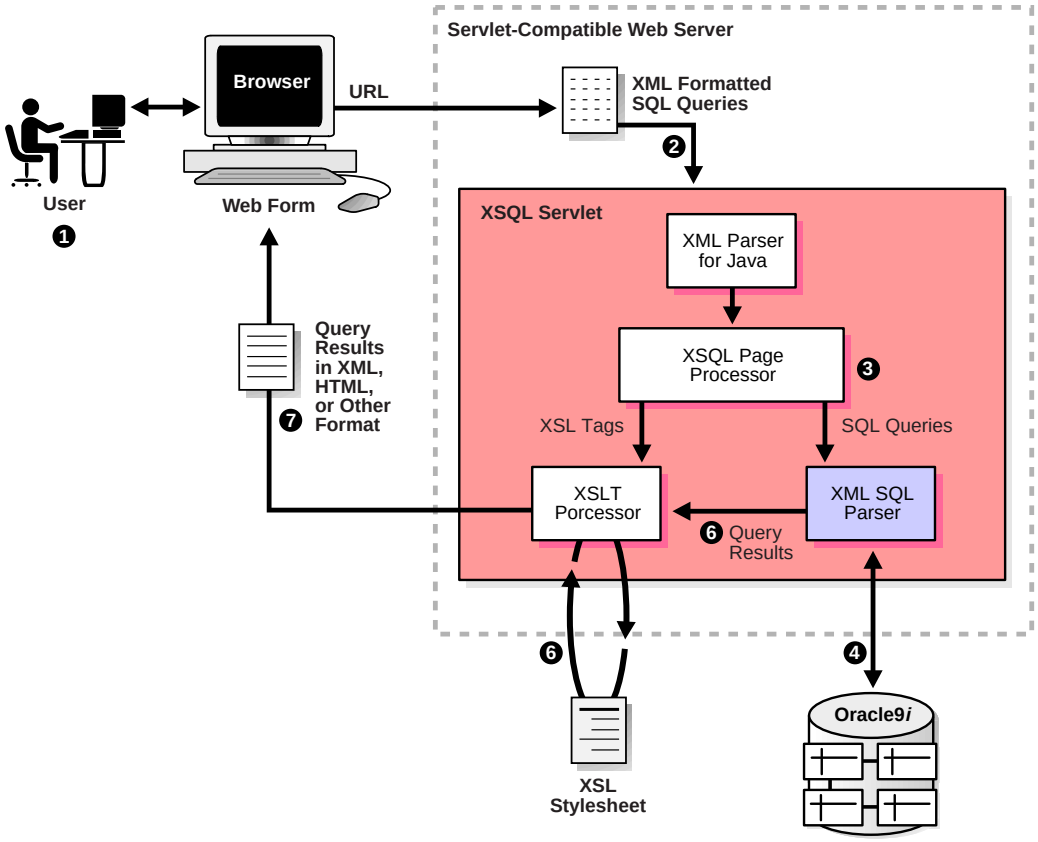
XSQL Servlet is a tool that processes SQL queries and outputs the result set as XML. This processor is implemented as a Java servlet and takes as its input an XML file containing embedded SQL queries. It uses XML Parser for Java and XML SQL Utility to perform many of its operations.

Figure 1–5 shows how data flows from a client, to the servlet, and back to the client. The sequence of events is as follows:

1. The user enters a URL through a browser, which is interpreted and passed to the XSQL Servlet through a Java Web Server. The URL contains the name of the target XSQL file (.xsql) and optionally, parameters, such as values and an XSL stylesheet name. Alternatively, the user can invoke the XSQL Servlet from the command line, bypassing the browser and Java web server.
2. The servlet passes the XSQL file to the XML Parser for Java, which parses the XML and creates an API for accessing the XML contents.
3. The page processor component of the servlet uses the API to pass XML parameters and SQL statements (found between `<query></query>` tags) to XML SQL Utility. The page processor also passes any XSL processing statements to the XSLT Processor.
4. XML SQL Utility sends the SQL queries to the underlying Oracle9*i* database, which returns the query results to the utility.
5. XML SQL Utility returns the query results to the XSLT Processor as XML formatted text. Results are embedded in the XML file in the same location as the original `<query>` tags.
6. If desired, the query results and any other XML data are transformed by the XSLT processor using a specified XSL stylesheet. The data can be transformed to HTML or any other format defined by the stylesheet. The XSLT processor can selectively apply different stylesheets based on the type of client that made the original URL request. This `HTTP_USER_AGENT` information is obtained from the client through an HTTP request.
7. The XSLT Processor passes the completed document back to the client browser for presentation to the user.

See Also: Chapter 9, "XSQL Pages Publishing Framework"

Figure 1–5 Oracle XSQL Page Processor and Servlet Functional Diagram



Oracle XML SQL Utility (XSU)

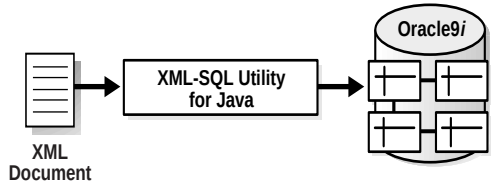
Oracle XML SQL Utility (XSU) supports Java and PL/SQL.

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

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

Figure 1–6 shows the Oracle XML SQL Utility overall functionality.

Figure 1–6 Oracle XML SQL Utility Functional Diagram



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

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

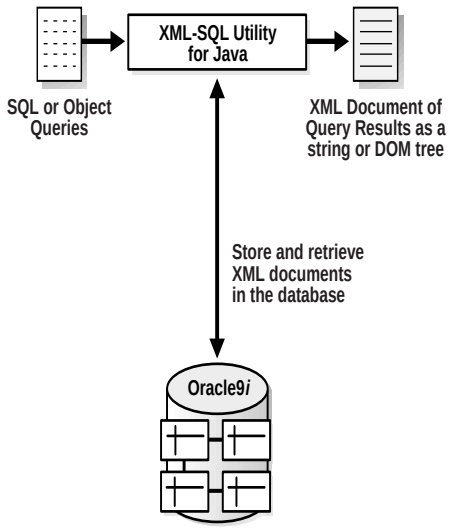
– Write XML data to a database table

See Also: Chapter 8, "XML SQL Utility (XSU)"

Generating XML from Query Results

Figure 1–7 shows how XML SQL Utility processes SQL queries and returns the results as an XML document.

Figure 1–7 XMl-SQL Utility Processes SQL Queries and Returns the Result as an XML Document



XML Document Structure: Columns Are Mapped to Elements

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

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

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

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

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

XSU Generates a DTD Based on Queried Table's Schema

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

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

> **See Also:** Chapter 8, "XML SQL Utility (XSU)" and *Oracle9i XML Case Studies and Applications*, the chapter, "B2B XML Application: Step by Step", for more information about this approach.

Note: To write an XML document to a database table, where the XML data does not match the underlying table structure, transform the XML document before writing it to the database. For techniques on doing this, see Chapter 8, "XML SQL Utility (XSU)".

TransX Utility

TransX Utility is a data transfer utility that enables you to populate your database with multilingual data. It uses XML to specify the data, so that you can take advantage of easy data transfer from XML to the database, a simple data format that is intuitive for both developers and translators, and validation capability that is less error prone than previous techniques.

See Also: Chapter 12, "Oracle TransX Utility"

Oracle Text

Oracle Text extends Oracle9*i* by indexing any text or documents stored in Oracle9*i*. Use Oracle Text to perform searches on XML documents stored in Oracle9*i* by indexing the XML as plain text, or as document sections for more precise searches, such as find `Oracle WITHIN title` where `title` is a section of the document.

See Also: For more information on using Oracle Text and XML, see:

- *Oracle Text Reference*
- *Oracle Text Application Developer's Guide*
- `http://otn.oracle.com/products/text`

XML Gateway

XML Gateway is a set of services that enables easy integration with the Oracle e-Business Suite to create and consume XML messages triggered by business events. It integrates with Oracle Advanced Queuing to enqueue/dequeue a message which is then transmitted to/from the business partner through any message transport agent.

See Also:

- *Oracle9i Application Developer's Guide - Advanced Queuing*
- *Oracle9i XML Database Developer's Guide - Oracle XML DB*

Oracle XML Components: Generating XML Documents

Figure 1–8 through Figure 1–11 illustrate the relationship of the Oracle XML components and how they work together to generate XML documents from Oracle9*i* through a SQL query. The options are depicted according to language used:

- Java
- C

- C++
- PL/SQL

Using Oracle XML Components to Generate XML Documents: Java

Figure 1–8 shows the Oracle XML Java components and how they can be used to generate an XML document. Available XML Java components are:

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

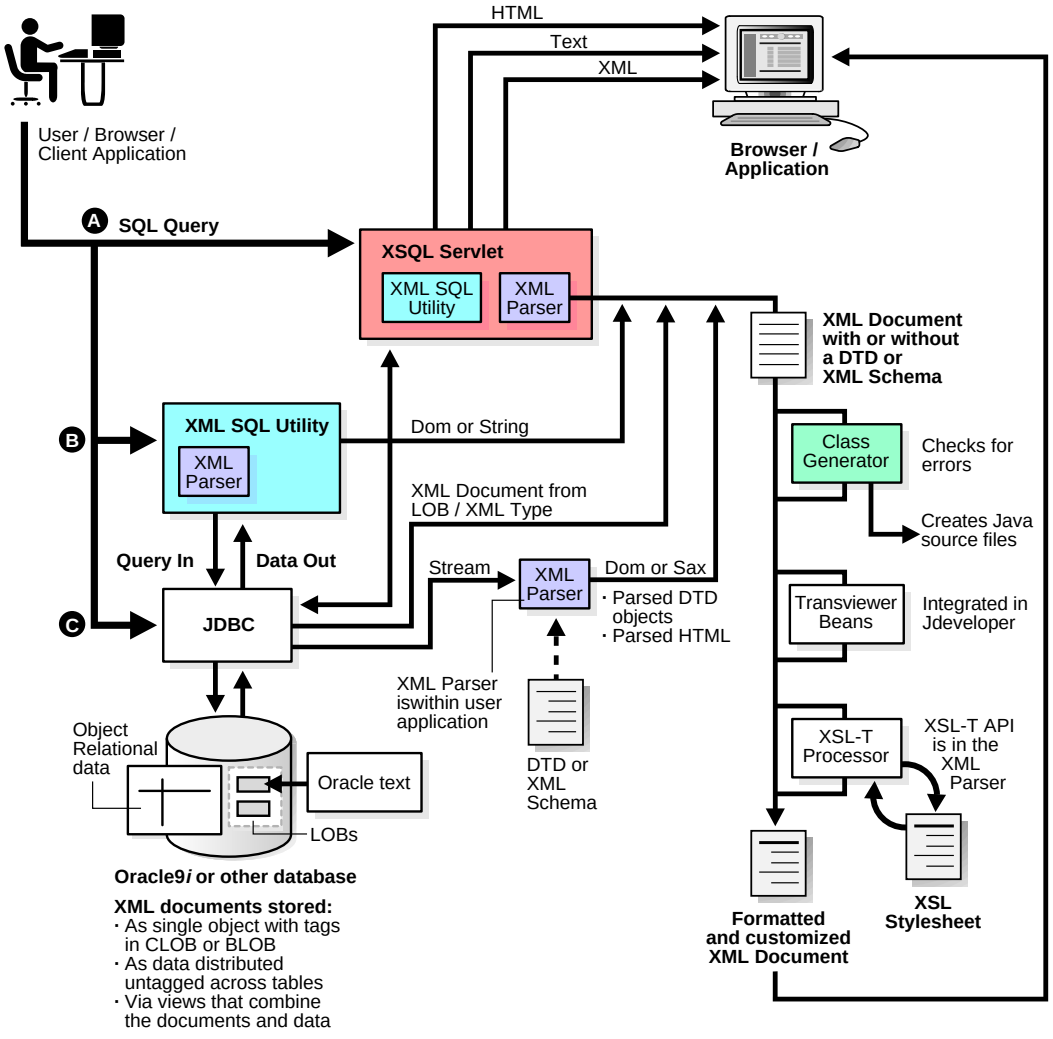
In the Java environment, when a user or client or application sends a query (SQL), there are three possible ways of processing the query using the Oracle XML components:

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

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

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

Figure 1–8 Generating XML Documents Using XDK for Java



Using Oracle XML Components to Generate XML Documents: C

Figure 1–9 shows the Oracle XML C language components used to generate an XML document. The XML components are:

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

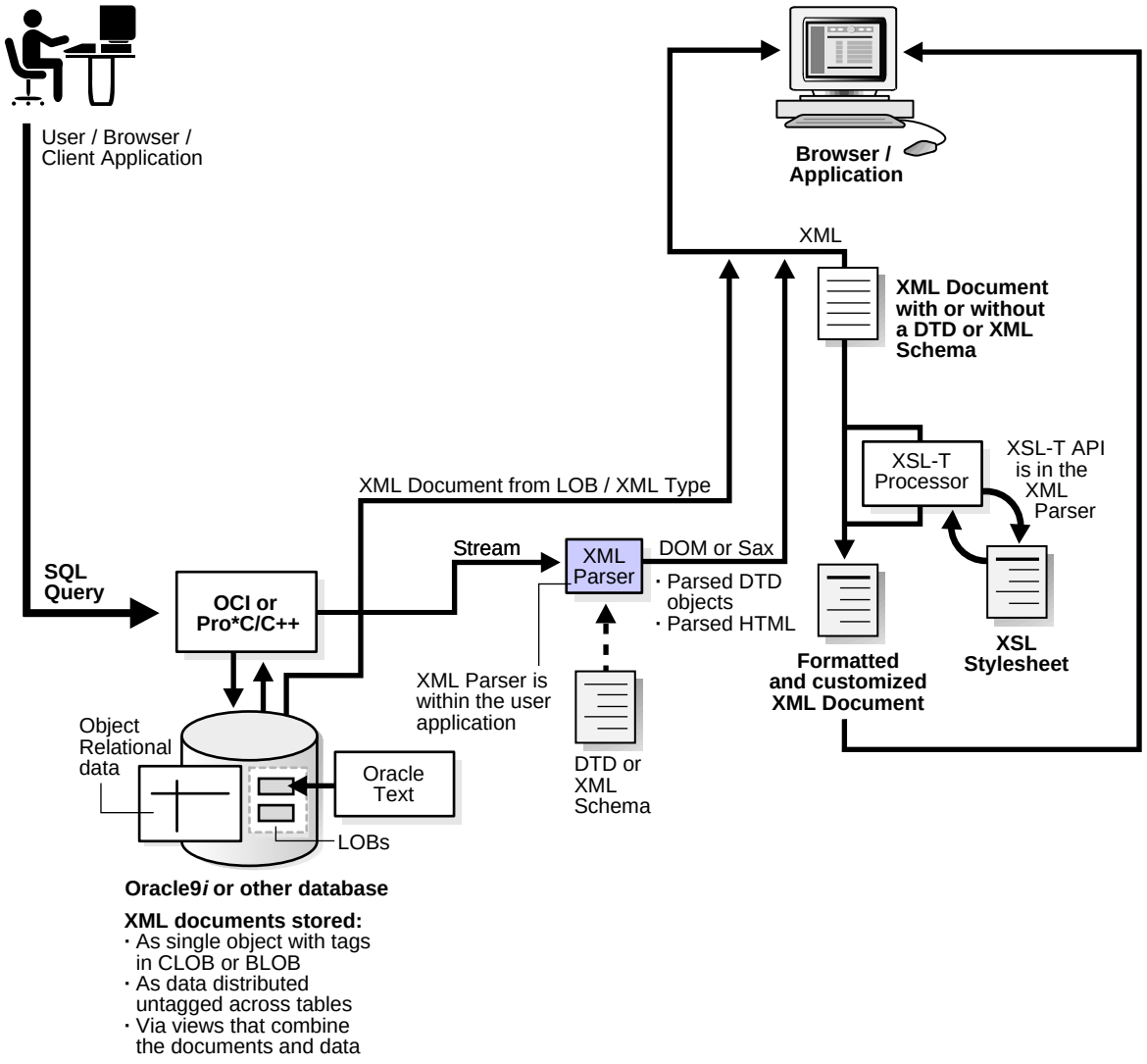
SQL queries can be sent to the database by OCI or as embedded statements in the Pro*C / C++ precompiler.

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

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

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

Figure 1–9 Generating XML Documents Using XDK for C



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

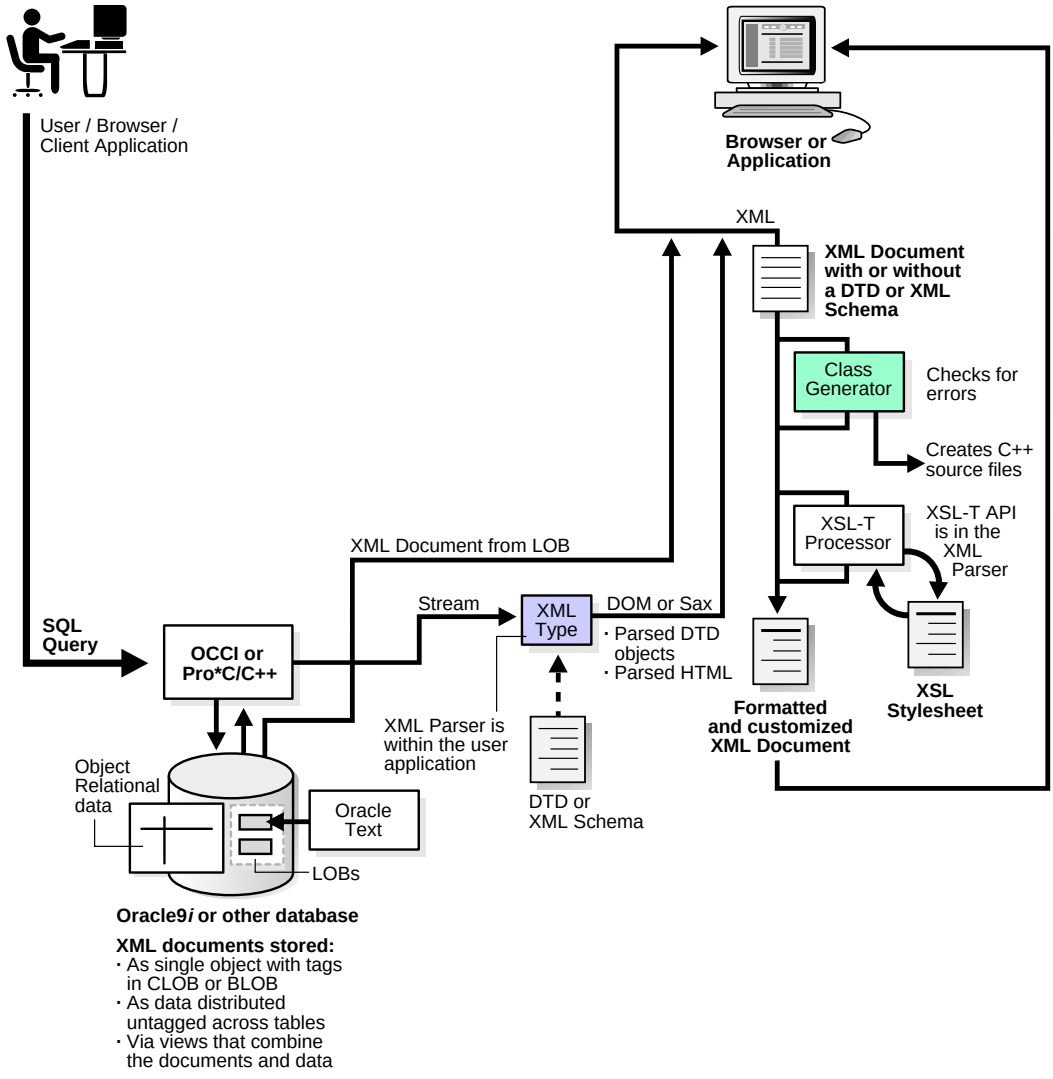
Figure 1–10 shows the Oracle XML components used to generate an XML document. The XDK for C++ components used here are:

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

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

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

Figure 1–10 Generating XML Documents Using XDK for C++



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

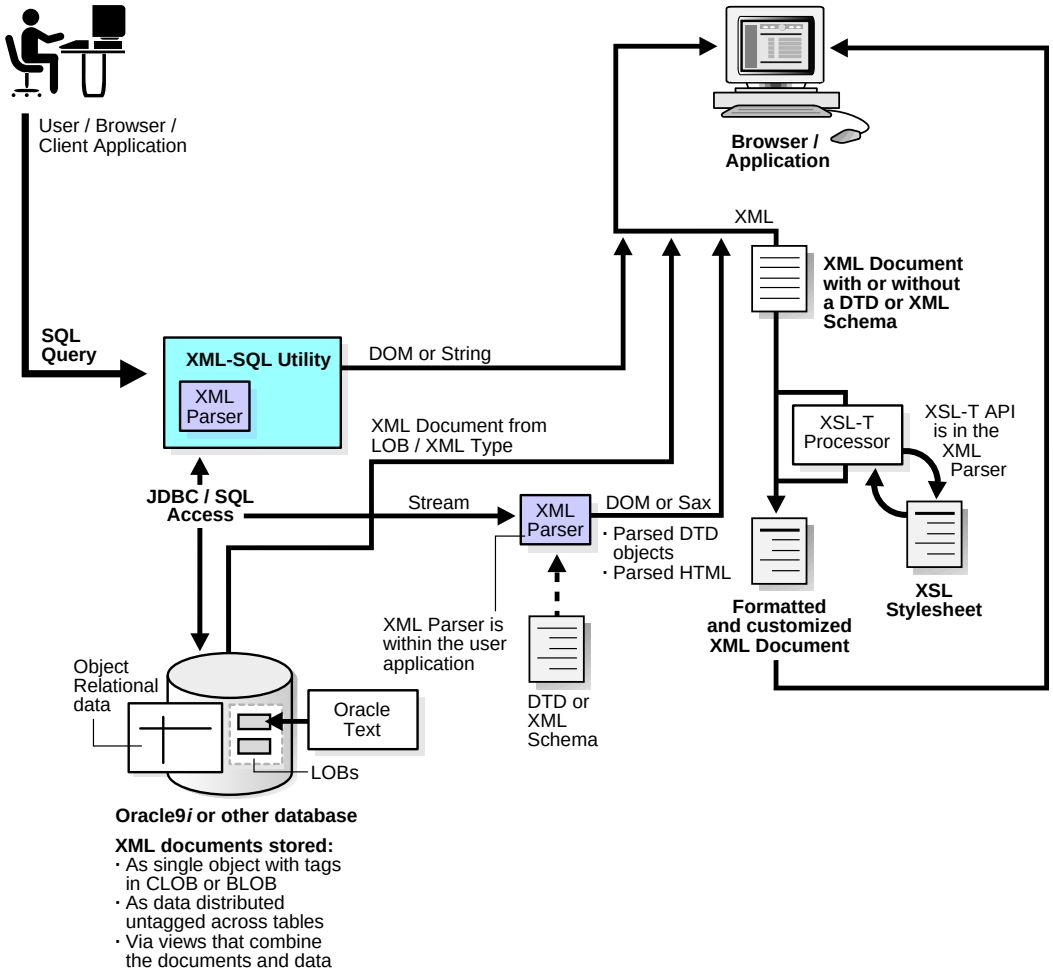
Figure 1–11 shows the XDK for PL/SQL components used to generate an XML document:

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

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

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

Figure 1–11 Generating XML Documents Using XDK for PL/SQL



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

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

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

Frequently Asked Questions About the XDK

What XML Components Do I Need to Install?

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

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

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

What Software Is Needed to Build an XML Application?

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

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

Answer:

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

XML Questions

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

1. What is the best way to design tables and generate XML flat tables, objects, or collections?

2. Can I use XML SQL Utilities in Pro*C/C++?
3. Is there a limiting size for generating XML documents from database?

Answer:

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

Are There XDK Utilities That Translate Data from Other Formats to XML?

I know that the XSLT will translate from XML to XML, HTML, or another text-based format. What about the other way around?

Answer: For HTML, you can use utilities like Tidy or JTidy to turn HTML into well-formed HTML that can be transformed using XSLT. For unstructured text formats, you can try utilities like XFlat at the following Web site:

`http://www.unidex.com/xflat.htm`.

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

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

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

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

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

Does Oracle have any tools for creating XML documents based on DTDs or the XML schema definition DOM, or for editing XML documents with DTD or schema validation?

Answer: JDeveloper9*i* has an integrated XML schema-driven code editor for working on XML schema-based documents such as XML schemas and XSLT stylesheets, with tag insight to help you easily enter the correct elements and attributes as defined by the schema.

See Also: Chapter 21, "XSLT Processor for PL/SQL"

How Can I Format XML Documents as PDF?

I have been asked to take stored XML docs in release 8.1.6 and format them as PDF. We are using JDeveloper release 3.1.1.2 as our development environment and the client wants to stick to OAS 4082 on Windows NT if possible. Any suggestions or recommended resources?

Answer: Oracle XSQL Pages release 1.0.2 supports integration with Apache FOP 0.14.0 for rendering PDF output from XML or SQL input.

It is possible to format XML into PDF using Formatting Object (FOP). See information on this at the following Web sites:

```
http://xml.apache.org/fop/
```

```
http://www.xml.com/pub/rg/75)
```

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

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

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

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

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

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

Can SQL*Loader Support Nesting?

If you have the following scenario:

```
...
     <something>
        <price>10.00</price>
     </something>
...
   ...
      ...
          <somethingelse>
            <price>55.00</price>
          </somethingelse>
```

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

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

`sample.xml`

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

```
</resultset>
```

`sample.ctl` -- field definition part of the SQL Loader control file

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

Keep in mind that the `COLUMN OBJECT` field names have to match the object column in the database. You will have to use a custom record terminator, otherwise it defaults to `newline` (that is, the `newline` separates data for a complete database record).

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

The SQL*Loader has a very powerful text parser. You can use it for loading XML when the document is very big.

Frequently Asked Questions About Previous Oracle Releases

Can I Use Parsers from Different Vendors?

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

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

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

Is There XML Support in Oracle Release 8.0.6?

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

Answer: All of our XML Developer's Kit components, including the XML Parser, XSLT Processor, XSQL Servlet, and utilities like the XML SQL Utility all work outside the database against Oracle 8.0.6. However, you will not be able to run XML components inside the database or use Oracle Text XML searching, which are both features in Oracle8*i* and higher.

Can I Do Data Transfers to Other Vendors Using XML from Oracle Release 7.3.4?

My company has Oracle release 7.3.4 and my group is thinking of using XML for some data transfers between us and our vendors. From what I could see from this Web site, it looks like we would need to move to Oracle8*i* or higher in order to do so. Is there any way of leveraging Oracle release 7 to do XML?

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

For JDBC drivers, refer to the following Web site for information about Oracle7 JDBC OCI and JDBC Thin Drivers:

`http://otn.oracle.com/tech/java/sqlj_jdbc/`

If I Use Versions Prior to Oracle8*i* Can I Use Oracle XML Tools?

If I am using an Oracle version earlier than Oracle8*i*, can I supply XML based applications using Oracle XML tools? If yes, then what are the licensing terms?

Answer: The Oracle XDKs for Java, C, and C++ can work outside the database, including the XML SQL Utility and XSQL Pages framework. Licensing is the same, including free runtime. See OTN for the latest licenses.

Can I Create Magnetic Tape Files with Oracle XML?

Is Oracle XML technology suitable for creating magnetic tape files where the file is just a string of characters like '`abcdefg.....`' in a particular format? Is it is possible to create a stylesheet that will create these kind of files?

Answer:

Yes. Just use `<xsl:output method="text"/>` to output plain text.

Frequently Asked Questions About Browsers that Support XML

Which Browsers Support XML?

Answer: The following browsers support the display of XML:

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

Frequently Asked Questions About XML Standards

Are There Advantages of XML Over EDI?

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

Answer: Here are some thoughts on the subject:

- EDI is a difficult technology: EDI enables machine-to-machine communication in a format that developers cannot easily read and understand.

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

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

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

What B2B Standards and Development Tools Does Oracle Support?

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

Answer: Oracle participates in several B2B standards organizations:

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

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

- Oracle Exchange delivers an out-of-the-box solution for implementing electronic marketplaces
- Oracle Integration Server (and primarily Message Broker) for in-house implementations
- Oracle Gateways for exchanges at data level

- Oracle XML Gateway to transfer XML-based messages from our e-business suite.

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

What Is Oracle Corporation's Direction Regarding XML?

Answer: Oracle Corporation's XML strategy is to use XML in ways that exploit all of the benefits of the current Oracle technology stack. Today you can combine Oracle XML components with the Oracle8*i* (or higher) database and Advanced Queueing (AQ) to achieve conflict resolution, transaction verification, and so on. Oracle is working to make future releases more seamless for these functions, as well as for functions such as distributed two phase commit transactions.

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

From Oracle9*i*, Oracle supports sending XML payloads using AQ. This involves making XML queriable from SQL.

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

What Is Oracle Corporation's Plans for XML Query?

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

Oracle has representatives participating actively in the following 3C Working Groups related to XML and XSL: XML Schema, XML Query, XSL, XLink / XPointer, XML Infoset, DOM, and XML Core.

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

We have implemented Oracle8*i* and the XDK. Where can we find basic, standard DTDs to build on for orders, shipments, and acknowledgements?

Answer: A good place to start would be this Web site, which has been set up for that purpose:

```
http://www.xml.org
```

Frequently Asked Questions About XML, CLOBs, and BLOBs

Is There Support for XML Messages in BLOBs?

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

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

Frequently Asked Questions About Maximum File Sizes

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

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

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

```
http://otn.oracle.com/tech/java/sqlj_jdbc/index2.htm?Code&files/advanced/advanced.htm
```

Are There Any Limitations on the Size of an XML File?

Answer: There are no XML limitations to an XML file size.

What Is the Maximum Size for an XML Document?

Is there a maximum size for an XML document to provide data for PL/SQL (or SQL) across tables, given that no CLOBs are used?

Also, what is the maximum size of XML document generated from Oracle to an XML document?

Answer:

The size limit for an XML document providing data for PL/SQL across tables should be what can be inserted into an object view.

The size limit for an XML document generated from Oracle to an XML document should be what can be retrieved from an object view.

Frequently Asked Questions About Inserting XML Data into Tables

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

To select data for display and insert data to tables by XML what software do I need? We are using Oracle8*i* on Solaris™ Operating Environment.

Answer: You need the following software:

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

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

- A Java compliant Web server
- XSQL Servlet

Frequently Asked Questions About XML Performance in the Database

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

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

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

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

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

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

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

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

Frequently Asked Questions About Multiple National Languages

How Do I Put Information in Chinese into XML?

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

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

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

The database should support all the languages you are going to use on XML. Chinese character sets such as ZHS16GBK and ZHT16BIG5 are a superset of ASCII

so you may be able to use one of them to serve for English and Chinese, but you may want to use Unicode to use more languages.

Frequently Asked Questions About Reference Material

Here are some other XML Frequently Asked Question sites of interest:

- `http://www.ucc.ie/xml/`
- `http://www.oasis-open.org/cover/`

What Are Some Recommended XML and XSL Books?

Answer:

- The publisher WROX has a number of helpful books. One of these, *XML Design and Implementation* by Paul Spencer, covers XML, XSL and development well.
- *Building Oracle XML Applications* by Steve Muench (published by O'Reilly) See `http://www.oreilly.com/catalog/orxmlapp/`
- *The XML Bible*. Read the updated chapter 14 from: `http://metalab.unc.edu/xml/books/bible/`

  far a good understanding of XSLT. Downloading this chapter is free.
- *Oracle9i XML Handbook* by the Oracle XML Product Development Team at `http://www.osborne.com/oracle/`

2

Getting Started with XDK for Java and JavaBeans

This chapter contains the following sections:

- Installation of the XDK for Java
- Installation of the XDK for JavaBeans

> **See Also:** Chapter 3, "Getting Started with XDKs for C/C++ and PL/SQL"

Installation of the XDK for Java

XDK for Java contains the basic building blocks for reading, manipulating, transforming and viewing XML documents.

> **Note:** The XDKs for Java and JavaBeans are now bundled together.

Oracle XDK for Java consists of the following components:

- XML Parser: supports parsing XML documents with both the DOM or SAX interfaces.
- XSL Processor: is included as part of the XML Parser and supports transforming XML documents.
- XML Schema Processor: supports parsing and validating XML files against an XML Schema definition file (default extension `.xsd`).
- Class Generator: generates a set of Java source files based on an input DTD or XML Schema.
- XML SQL Utility: generates an XML Document from SQL queries and inserts the document into the database.
- TransX Utility: makes it easier to load translated seed data and messages into the database.
- XSQL Servlet: produces dynamic XML documents based on one or more SQL queries.

Installation Steps for XDK for Java

XDK for Java comes with the Oracle database and with the application server. Or, you can download the latest beta or production version of XDK for Java from OTN.

If you installed XDK with Oracle database or iAS, you can skip the following steps and change into the XDK home directory ($XDK_HOME).

If you need to download the XDK from OTN, follow these steps:

Go to the URL:

`http://otn.oracle.com/tech/xml/xdk_java/content.html`

Click on the ‘Software’ icon at the left side of the page.

- Logon with your OTN username and password (registration is free if you do not already have an account).
- Select the version you want to download.
- Accept all terms of the licensing agreement and download the software. Here are the instructions found on the download site for Solaris™ Operating Environment:

```
Oracle XML Developer's Kit for Java on Sun Solaris™ Operating Environment- 9i
   Download the Complete File

   xdk_java_9_0_1_1_0A.tar.gz
   Directions

   Install  GNU gzip.

   Download the Oracle XDK for Java in .tar format

   Extract the distribution package into a directory.
   (Ex: #gzip -dc xdk_java.tar | tar xvf -)

   The result should be the following files and directories:
         /bin - xdk executables and utilities
         /lib - directory for libraries
         /xdk - top xdk directory
         /xdk/demo - directory for demo files
         /xdk/doc - directory for documentation
         /xdk/admin - direcory for dband config files
         /xdk/*html. - doc navigation files
         /xdk/license.html - copy of license agreement
```

- For Windows NT, choose a directory under which you would like the .\xdk directory and subdirectories to go (for example, C:\ on NT), change the directory to C:\ then extract the files using the WinZip visual tool.

What Are the XDK for Java Components?

After installing the XDK, the directory structure is:

```
-$XDK_HOME
    | - bin: executable files and setup script/batch files.
    | - lib: library files.
    | - xdk:
         | - admin: (Administration): XSU PL/SQL API setup SQL script
```

```
        and XSL Servlet Configuration file(XSQLConfig.xml).
| - demo: demonstration code
| - doc: documents including release notes and javadocs.
```

All the packages in XDK for Java are certified and supported with JDK 1.2 or JDK 1.1.8, so make sure that your CLASSPATH includes all the necessary libraries:

Table 2–1 XDK for Java Libraries

Component	Library	Notes
XML Parser XSL Processor	xmlparserv2.jar	XML Parser V2 for Java, which includes JAXP 1.1, DOM, SAX and XSLT APIs.
	xmlmesg.jar	Message files for XML Parser. If you want to use XML Parser with a language other than English, you need to set this JAR file in your CLASSPATH.
XML Schema Processor	xschema.jar	XML Schema Processor for Java
XML SQL Utility	xsu12.jar	XML SQL Utility for JDK 1.2 and above
	xsu111.jar	XML SQL Utility for JDK 1.1.1
XSQL Servlet	oraclesql.jar	Oracle XSQL Servlet
	xsqlserializers.jar	Oracle XSQL Serializers for FOP / PDF Integration
	classgen.jar	XML Class Generator for Java
	transx.zip	Oracle TransX Utility

In addition, XML SQL Utility, XSQL Servlet and TransX Utility all depend on JDBC, which is listed in the following table:

Table 2–2 XDK Libraries for Java

Component	Library	Notes
JDBC	classes12.zip	JDBC for JDK 1.2 and above
	classes111.zip	JDBC for JDK 1.1.8
Globalization	nls_charset12.jar	Globalization support for JDK 1.2 and above
	nls_charset111.jar	Globalization support for JDK 1.1.8

Environment Settings for XDK for Java

These files will set up the environment:

UNIX: `$XDK_HOME/bin/env.csh`

NT: `$XDK_HOME/bin/env.bat`

The following tables list the environment variables, with the ones that must be customized marked with "Y":

Table 2–3 NT Environment Settings

Variable	Notes	Y/N
%JDBCVER%	Directory where the Java™ 2 SDK, Standard Edition, version 1.3.1 is installed	Y
%JDKVER%	Include the following: .;%XDK_HOME%\lib\xmmlparserv2.jar;%XDK_HOME%\lib\xsu12.jar;	Y
%INSTALL_ROOT%	Installation root of XDK which is the directory we refer to as %XDK_HOME%.	N
%JAVA_HOME%	JAVA_HOME=C:\JDK%JDKVER%	Y
%CLASSPATHJ%	CLASSPATHJ=%ORACLE_HOME%\jdbc\lib\classes%JDBCVER%.zip; %ORACLE_HOME%\jdbc\lib\nls_charset%JDBCVER%.jar	Y
%PATH%	PATH=%JAVA_HOME%\bin;%ORACLE_HOME%\bin;%PATH%;%INSTALL_ROOT%\bin	N
%CLASSPATH%	.;%CLASSPATHJ%;%INSTALL_ROOT%\lib\xmlparserv2.jar; %INSTALL_ROOT%\lib\xschema.jar; %INSTALL_ROOT%\lib\xsu%JDBCVER%.jar; %INSTALL_ROOT%\lib\oraclexsql.jar;%INSTALL_ROOT%\lib\classgen.jar	N

The following table shows the UNIX environment variables (the ones that must be customized are marked with "Y"):

Table 2–4 UNIX Environment Settings

Variable	Notes	Y/N
$JDBCVER	JDBC Version. If using JDK 1.2 and above, it should be set to 12. If using JDK 1.1.8, it should be set to 111	Y
$JDKVER	JDK Version which you can get from: Java -version For example, the default value is: 1.2.2_07	Y
$INSTALL_ROOT	Installation root of XDK, which is the directory referred to as $XDK_HOME.	N
$JAVA_HOME	Directory where the Java SDK, Standard Edition is installed.	Y
$CLASSPATHJ	Path linked to the Java SDK needs to be modified. ${ORACLE_HOME}/jdbc/lib/classes${JDBCVER}.zip: ${ORACLE_HOME}/jdbc/lib/nls_charset${JDBCVER}.jar If you are running the XSU on a system different then where the Oracle RDBMS is installed, you will have to update CLASSPATHJ path with the correct locations of the JDBC library (classes12.jar). The nls_charset12.jar is needed to support certain character sets. Refer to Globalization setup with XDK for Java Note that if you don't have these libraries on your system, these are both available on OTN (http://otn.oracle.com) -- part of JDBC driver download	Y
$CLASSPATH	Include the following: .:${CLASSPATHJ}:${INSTALL_ROOT}/lib/xmlparserv2.jar: ${INSTALL_ROOT}/lib/xschema.jar: ${INSTALL_ROOT}/lib/xsu${JDBCVER}.jar: ${INSTALL_ROOT}/lib/oraclexsql.jar: ${INSTALL_ROOT}/lib/classgen.jar	N
$PATH	${JAVA_HOME}/bin:${PATH}:${INSTALL_ROOT}/bin	N
$LD_LIBRARY_PATH	For OCI JDBC connections. ${ORACLE_HOME}/lib:${LD_LIBRARY_PATH}	N

XSU Setup

XSU installation is discussed in "Installation of XDK for PL/SQL" on page 3-25.

XSQL Servlet Setup

The XSQL Servlet is designed to run on any Java VM, using any JDBC driver, against any database. In practice, we are able to test it against only the most popular configurations; we document the supported configurations that have been tested in the Oracle labs.

XSQL Pages and XSQL Servlet have been successfully tested only with:

- JDK 1.1.8
- JDK 1.2.2
- JDK 1.3

These are the only three JDK versions that we *know* work correctly.

> **Note:** Numerous users have reported problems using XSQL Pages and XSQL Servlet with JDK 1.1.7. These problems are in the character set conversion routines for UTF-8 and make JDK 1.1.7 unusable for processing XSQL Pages.

Supported Servlet Engines

This XSQL Servlet has been tested with the following servlet engines:

- Oracle9iAS Apache / JServ Servlet Engine
- Oracle9iAS OC4J Servlet Engine
- Allaire JRun 2.3.3 and 3.0.0
- Apache 1.3.9 with JServ 1.0 and 1.1
- Apache 1.3.9 with Tomcat 3.1 or 3.2 Servlet Engine
- Apache Tomcat 3.1 or 3.2 Web Server + Servlet Engine
- Caucho Resin 1.1
- Java Web Server 2.0
- Weblogic 5.1 Web Server
- NewAtlanta ServletExec 2.2 and 3.0 for IIS / PWS 4.0
- Oracle8*i* Lite Web-to-Go Server
- Oracle8*i* 8.1.7 Oracle Servlet Engine

- Sun JavaServer Web Development Kit (JSWDK) 1.0.1 Web Server

Supported JSP Implementations

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

- Oracle9iAS Apache / JServ Servlet Engine
- Oracle9iAS OC4J Servlet Engine
- Apache 1.3.9 with Tomcat 3.1 or 3.2 Servlet Engine
- Apache Tomcat 3.1 or 3.2 Web Server + Tomcat 3.1 or 3.2 Servlet Engine
- Caucho Resin 1.1 (Built-in JSP 1.0 Support)
- NewAtlanta ServletExec 2.2 and 3.0 for IIS / PWS 4.0 (Built-in JSP 1.0 Support)
- Oracle8*i* Lite Web-to-Go Server with Oracle JSP 1.0
- Oracle8*i* 8.1.7 Oracle Servlet Engine
- Any Servlet Engine with Servlet API 2.1+ and Oracle JSP 1.0

In general, it should work with any servlet engine supporting the Servlet 2.1 Specification or higher, and the Oracle JSP 1.0 reference implementation or functional equivalent from another vendor.

JDBC Drivers and Databases

The Oracle XSQL Page processor has been designed to exploit the maximum set of features against the Oracle JDBC drivers, but gracefully works against any database with a reasonable JDBC driver. While numerous users have reported successfully using XSQL Pages with many other JDBC drivers, the ones that we have tested in-house are:

- Oracle8*i* 8.1.5 Driver for JDBC 1.*x*
- Oracle8*i* 8.1.6 Driver for JDBC 1.*x*
- Oracle8*i* 8.1.7 Driver for JDBC 1.*x*
- Oracle8*i* Lite 4.0 Driver for JDBC 1.x
- Oracle8*i* 8.1.6 Driver for JDBC 2.0
- Oracle8*i* 8.1.7 Driver for JDBC 2.0
- Oracle9*i* 9.0.1 Driver for JDBC 2.0

Setting Up the Database Connection Definitions for Your Environment

The demos are set up to use the SCOTT schema on a database on your local machine (the machine where the web server is running). If you are running a local database and have a SCOTT account whose password is TIGER, then you are all set. Otherwise, you need to edit the `.\xdk\admin\XSQLConfig.xml` file to correspond to your appropriate values for username, password, dburl, and driver values for the connection named demo:

```
<?xml version="1.0" ?>
<XSQLConfig>
     :
  <connectiondefs>
    <connection name="demo">
      <username>scott</username>
      <password>tiger</password>
      <dburl>jdbc:oracle:thin:@localhost:1521:ORCL</dburl>
      <driver>oracle.jdbc.driver.OracleDriver</driver>
    </connection>
    <connection name="lite">
      <username>system</username>
      <password>manager</password>
      <dburl>jdbc:Polite:POlite</dburl>
      <driver>oracle.lite.poljdbc.POLJDBCDriver</driver>
    </connection>
  </connectiondefs>
     :
</XSQLConfig>
```

Setting Up Your Servlet Engine to Run XSQL Pages

UNIX users and any user wanting to install the XSQL Servlet on other Web servers should continue with the instructions below depending on the Web server you're trying to use. In every case, there are these basic steps:

1. Include the list of XSQL Java archives:
   - `xsu12.jar` - Oracle XML SQL Utility
   - `xmlparserv2.jar` - Oracle XML Parser for Java V2
   - `oraclexsql.jar`- Oracle XSQL Pages
   - `xsqlserializers.jar` - Oracle XSQL Serializers for FOP / PDF Integration

- `classes12.jar` - Oracle JDBC Driver *or the JAR file for the JDBC driver you will be using instead*
- Include as well as the directory where XSQLConfig.xml resides (by default `./xdk/admin`) in the server `CLASSPATH`.

2. Map the `.xsql` file extension to the `oracle.xml.xsql.XSQLServlet` servlet class.
3. Map a virtual directory `/xsql` to the directory where you extracted the XSQL files (to access the online help and demos).

Oracle Internet Application Server Oracle IAS release 1.0 and higher comes preconfigured to run XSQL Servlet. By default its Apache JServ servlet engine contains all of the `wrapper.classpath` entries in `jserv.conf` to include the necessary Java archives to run XSQL. The `XSQLConfig.xml file` is found in the `./xdk/admin` subdirectory of the IAS installation home.

Oracle 9iAS Oracle Containers for Java (OC4J) Servlet Container The easiest way to install XSQL Servlet in the Oracle9iAS OC4J servlet container is to install it as a global application. Assuming your OC4J installation home is `C:\j2ee\home`, and that you've extracted the XDK distribution into the `C:\xdk902` directory, here are the setup steps:

1. Verify that the following JAR files are already in your `C:\j2ee\home\lib` directory (they should come pre-installed):
   - `xmlparserv2.jar` - Oracle XML Parser for Java V2
   - `classes12.jar` - Oracle JDBC Driver
2. Copy the following additional JAR files from `C:\xdk902\lib` to `C:\j2ee\home\lib`.
   - `xsu12.jar` - Oracle XML SQL Utility
   - `oraclexsql.jar` - Oracle XSQL Pages
   - `xsqlserializers.jar` - Oracle XSQL Serializers for FOP / PDF Integration
3. Copy the `C:\xdk\admin\XSQLConfig.xml` configuration file to the `C:\j2ee\home\default-web-app\WEB-INF\classes` directory.
4. Edit the `C:\j2ee\home\config\global-web-application.xml server` configuration file to add a `<servlet>` and `<servlet-mapping>` entry as child elements of the `<web-app>` element as follows:

```
<orion-web-app ...and so on... >
  :
 etc
  :
  <web-app>
    <servlet>
      <servlet-name>xsql</servlet-name>
      <servlet-class>oracle.xml.xsql.XSQLServlet</servlet-class>
    </servlet>
    <servlet-mapping>
      <servlet-name>xsql</servlet-name>
      <url-pattern>/*.xsql</url-pattern>
    </servlet-mapping>
     :
    etc
     :
  </web-app>
</web-app>
```

At this point, you can refer to any XSQL page in any virtual path and it will be processed by the XSQL Servlet. If you want to try the XSQL built-in samples, demos, and online help, then you need to perform the following additional step to map a virtual path of `/xsql/` to the `C:\xdk\demo\java\xsql` directory.

Edit the file:

```
c:\j2ee\home\application-deployments\default\defaultWebApp\orion-web.xml
```

to add the following `<virtual-directory>` entry:

```
<orion-web-app ...and so on...>
    :
   etc
    :
   <virtual-directory
   virtual-path="/xsql"
   real-path="/c:/xdk/xdk/demo/java/xsql/" />
    :
   etc
    :
</orion-web-app>
```

Then, you can browse the demos using the URL:

```
http://yoursever:yourport/xsql/index.html
```

Apache JServ 1.0 or 1.1 Setup the server CLASSPATH correctly for the XSQL Servlet. This is done by editing the JServ configuration file named `jserv.properties`. Assuming you installed the XSQL Servlet files into C:\, you need to add the following entries to use the Oracle JDBC 1.x Driver:

```
# Oracle XML SQL Utility (XSU)
wrapper.classpath=C:\xdk902\lib\xsu111.jar
# Oracle XSQL Servlet
wrapper.classpath=C:\xdk902\lib\oraclexsql.jar
# Oracle JDBC (8.1.6) -- JDBC 1.x driver
wrapper.classpath=directory_where_JDBC_Driver_resides\classes111.zip
# Oracle XML Parser V2 (with XSLT Engine)
wrapper.classpath=C:\xdk902\lib\xmlparserv2.jar
# XSQLConfig.xml File location
wrapper.classpath=directory_where_XSQLConfig.xml_resides
# FOR Apache FOP Generation, Add
# wrapper.classpath=C:\xdk902\lib\xsqlserializers.jar
# wrapper.classpath=FOPHOME/fop.jar
# wrapper.classpath=FOPHOME/lib/batik.jar
```

To use the Oracle JDBC 2.0 Driver, the list looks like:

```
# Oracle XML SQL Utility (XSU)
wrapper.classpath=C:\xdk902\lib\xsu12.jar
# Oracle XSQL Servlet
wrapper.classpath=C:\xdk902\lib\oraclexsql.jar
# Oracle JDBC (8.1.6) -- JDBC 2.0 driver
wrapper.classpath=directory_where_JDBC_Driver_resides\classes12.zip
# Oracle XML Parser V2 (with XSLT Engine)
wrapper.classpath=C:\xdk902\lib\xmlparserv2.jar
# XSQLConfig.xml File location
wrapper.classpath=directory_where_XSQLConfig.xml_resides
# FOR Apache FOP Generation, Add
# wrapper.classpath=C:\xdk902\lib\xsqlserializers.jar
# wrapper.classpath=FOPHOME/fop.jar
# wrapper.classpath=FOPHOME/lib/w3c.jar
```

Map the .xsql file extension to the XSQL Servlet To do this, you need to edit the JServ configuration file named `jserv.conf` (in JServ 1.0 this was named `mod_jserv.conf` on some platforms). Add the following lines:

```
# Executes a servlet passing filename with proper extension in PATH_TRANSLATED
# property of servlet request.
# Syntax: ApJServAction [extension] [servlet-uri]
# Defaults: NONE
```

```
ApJServAction .xsql /servlets/oracle.xml.xsql.XSQLServlet
```

Map an /xsql/ virtual directory In this step, we want to map the virtual path `\xsql\` to `C:\xdk902\xdk\demo\java\xsql\` (or wherever you installed the XSQL Servlet files). To do this, you need to edit the Apache configuration file named `httpd.conf` and add the following line:

```
Alias /xsql/ "C:\xdk902\xdk\demo\java\xsql\"
```

Restart the Apache server and browse the URL:

```
http://localhost/xsql/index.html
```

Jakarta Tomcat 3.1 or 3.2

Set up the Server CLASSPATH for the XSQL Servlet This is done by editing the Tomcat startup script named `tomcat.bat` in `./jakarta-tomcat/bin` and adding five lines to append the appropriate entries onto the system `CLASSPATH` before the Tomcat server is started as shown below:

For Oracle JDBC 1.x Driver:

```
rem Set up the CLASSPATH that we need

set cp=%CLASSPATH%

set CLASSPATH=.
set CLASSPATH=%TOMCAT_HOME%\classes
set CLASSPATH=%CLASSPATH%;%TOMCAT_HOME%\lib\webserver.jar
set CLASSPATH=%CLASSPATH%;%TOMCAT_HOME%\lib\jasper.jar
set CLASSPATH=%CLASSPATH%;%TOMCAT_HOME%\lib\xml.jar
set CLASSPATH=%CLASSPATH%;%TOMCAT_HOME%\lib\servlet.jar
set CLASSPATH=%CLASSPATH%;%JAVA_HOME%\lib\tools.jar

REM Added for Oracle XSQL Servlet
REM ----------------------------
set CLASSPATH=%CLASSPATH%;C:\xdk902\lib\xsu111.jar
set CLASSPATH=%CLASSPATH%;C:\xdk902\lib\oraclexsql.jar
set CLASSPATH=%CLASSPATH%;C:\xdk902\lib\xmlparserv2.jar
set CLASSPATH=%CLASSPATH%;directory_where_JDBC_Driver_resides\classes111.zip
set CLASSPATH=%CLASSPATH%;directory_where_XSQLConfig.xml_resides
REM FOR Apache FOP Generation, Add
REM set CLASSPATH=%CLASSPATH%;C:\xdk902\lib\xsqlserializers.jar
REM set CLASSPATH=%CLASSPATH%;FOPHOME/fop.jar
REM set CLASSPATH=%CLASSPATH%;FOPHOME/lib/batik.jar
```

For Oracle JDBC 2.0 Driver:

```
rem Set up the CLASSPATH that we need

set cp=%CLASSPATH%

set CLASSPATH=.
set CLASSPATH=%TOMCAT_HOME%\classes
set CLASSPATH=%CLASSPATH%;%TOMCAT_HOME%\lib\webserver.jar
set CLASSPATH=%CLASSPATH%;%TOMCAT_HOME%\lib\jasper.jar
set CLASSPATH=%CLASSPATH%;%TOMCAT_HOME%\lib\xml.jar
set CLASSPATH=%CLASSPATH%;%TOMCAT_HOME%\lib\servlet.jar
set CLASSPATH=%CLASSPATH%;%JAVA_HOME%\lib\tools.jar

REM Added for Oracle XSQL Servlet
REM ----------------------------
set CLASSPATH=%CLASSPATH%;C:\xdk902\lib\xsu12.jar
set CLASSPATH=%CLASSPATH%;C:\xdk902\lib\oraclexsql.jar
set CLASSPATH=%CLASSPATH%;C:\xdk902\lib\xmlparserv2.jar
set CLASSPATH=%CLASSPATH%;directory_where_JDBC_Driver_resides\classes12.zip
set CLASSPATH=%CLASSPATH%;directory_where_XSQLConfig.xml_resides
REM FOR Apache FOP Generation, Add
REM set CLASSPATH=%CLASSPATH%;C:\xdk902\lib\xsqlserializers.jar
REM set CLASSPATH=%CLASSPATH%;FOPHOME/fop.jar
REM set CLASSPATH=%CLASSPATH%;FOPHOME/lib/batik.jar
```

Map the .xsql File Extension to the XSQL Servlet Tomcat supports creating any number of configuration *contexts* to better organize the web applications your site needs to support. Each context is mapped to a virtual directory path, and has its own separate servlet configuration information. XSQL Servlet comes with a preconfigured context file to make XSQL Servlet setup easier.

By default, Tomcat 3.1 and 3.2 come preconfigured with the following contexts (defined by `<Context>` entries in the `./jakarta-tomcat/conf/server.xml file`).

- The *root* context
- /examples
- /test
- /admin

We could install XSQL Servlet into one of these, but for simplicity we'll create a new context just for the XSQL Servlet that maps to the directory where you installed the XSQL Servlet distribution.

Edit the `./jakarta-tomcat/conf/server.xml` file to add the following `<Context>` entry with `path= "/xsql"`.

```
<Context path="/test" docBase="webapps/test" debug="0" reloadable="true" />

<!--
 |  Define a Servlet context for the XSQL Servlet
 |
 |  The XSQL Servlet ships with a .\WEB-INF directory
 |  with its web.xml file preconfigured for C:\xdk902\xdk\demo\java\xsql
 |  installation.
 +-->
<Context path="/xsql" docBase="C:\xdk902\xdk\demo\java\xsql"/>
```

Note that the docBase= "C:\xsql" points to the physical directory where you installed the XSQL Servlet distribution. You then need to create a WEB-INF subdirectory in the `C:\xdk902\xdk\demo\java\xsql` directory and save the following `./WEB-INF/web.xml` file in it:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.2//EN" "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
   <servlet>
      <servlet-name>oracle-xsql-servlet</servlet-name>
      <servlet-class>oracle.xml.xsql.XSQLServlet</servlet-class>
   </servlet>
    <servlet-mapping>
      <servlet-name>oracle-xsql-servlet</servlet-name>
      <url-pattern> *.xsql </url-pattern>
   </servlet-mapping>
</web-app>
```

Note: To add the XSQL Servlet to an existing context, add the servlet and servlet-mapping entries that you find in the `web.xml` file preceding, into the `web.xml` file for the context in question.

Map an /xsql/ Virtual Directory This is already achieved by creating the /xsql context preceding.

Restart the Tomcat server and browse the URL:

```
http://localhost:8080/xsql/index.html
```

If you use Tomcat with an XML Parser (such as the Sun Crimson Parser) that only supports DOM Level 1 interfaces, then you must edit `tomcat.bat` to insure that the Oracle XML Parser's archive `xmlparser.jar` comes before the DOM Level 1 parser's archive in the `CLASSPATH`. For example, you could edit `tomcat.bat` to add the following lines:

```
REM NEED TO PUT xmlparserv2.jar FIRST before parser.jar
set CP=C:\xdk902\lib\xmlparserv2.jar;%CP%
```

just before the lines:

```
echo Using CLASSPATH: %CP%
echo.
set CLASSPATH=%CP%
```

XDK for Java with Globalization Support

Here is a summary on the setting that related to Globalization Support.

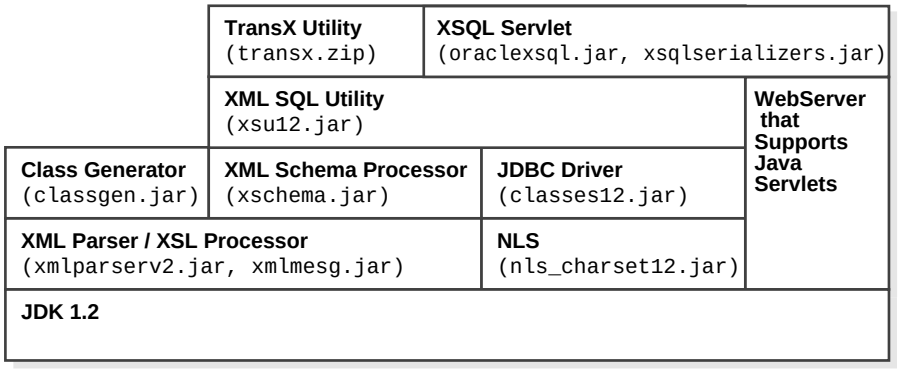
- Using `xmlmesg.jar`: If you are using the language other than English you would need to set the `xmlmesg.jar` into your `CLASSPATH` to let the parser get correct messages in your language.
- Using `nls_charset12.jar`: If you are using a multibyte character set other than one of the following,
  - UTF-8
  - ISO8859-1
  - JA16SJIS

  then you must set this JAR file into your Java `CLASSPATH` so that JDBC can convert the character set of the input file to the database character set during the loading of XML files using either XSU, TransX or XSQL Servlet.

XDK Dependencies

The following figure shows the dependencies of XDK when using JDK 1.2 and higher:

Figure 2–1 XDK Dependencies Using JDK 1.2.x and Higher



After you correctly setup the environment, include all the necessary JAR files in your CLASSPATH. You can then start writing your Java programs and compiling them with the `javac` command:

```
javac your_program.java
```

If the compilation finishes without errors, then you can just test your program using the command line or the Web Server.

> **See Also:** Chapter 4, "XML Parser for Java" for further discussion of the XDK for Java components

Installation of the XDK for JavaBeans

The XDK for JavaBeans permit easily adding visual or non-visual interfaces to XML applications. The bean encapsulation includes documentation and descriptors that can be accessed directly from Java Integrated Development Environments like JDeveloper.

> **Note:** The XDKs for Java and JavaBeans are now bundled together.

Oracle XDK for JavaBeans consists of the following components:

- DOMBuilder Bean encapsulates the DOMParser and provides asynchronous XML document parsing.

- TreeViewer Bean displays XML formatted files graphically as a tree. These branches and leaves of this tree can be manipulated with a mouse.
- SourceViewer Bean displays XML and XSL formatted files with color syntax highlighting for easy viewing and editing.
- Transformer Bean accepts an input XML document and applies the transformation specified by an input XSL stylesheet to create an output file.
- TransPanel Bean encapsulates the preceding beans in an application component for retrieving, transforming, and viewing XML files.
- DBAccess Bean can be used for programmatic access to all features that XMLTransformPanel offers in interactive mode with support of `XMLType`.
- DBView Bean can be used in any application that requires visualization of database information using XML and stylesheet transformations.
- XMLDiff Utility can be used to compare two XML files and represent the difference visually, or by the generated XSL code.
- XMLCompression Utility: can used to serialize XML document in Compressed format.

XDK for JavaBeans comes with Oracle Database or iAS application server. You can also download the latest versions of XDK for JavaBeans from OTN.

If you installed XDK with Oracle Database or iAS application server, you can skip the following steps and direct to refer to the XDK home directory (we will refer to this directory as `$XDK_HOME`).

If you need to download the XDK from OTN, follow these steps:

Use this URL in your browser:

```
http://otn.oracle.com/tech/xml/xdk_jbeans/index.html
```

Click on the `Software` icon at the left-hand side of the page.

- Log in with your OTN username and password (registration is free if you don not already have an account).
- Select the version you want to download.
- Accept all the terms of the licensing agreement and download the software. Here are the instructions found on the download site for Solaris™ Operating Environment:

  ```
  Oracle XML Developer's Kit for Java on Sun Solaris™ Operating Environment-
  9i
  ```

```
Download the Complete File
xdk_java_9_0_2_0_0C.tar.gz

Directions

      Install GNU gzip.
      Download the Oracle XDK for JavaBeans in .tar format
      Extract the distribution package into a directory.

(Ex: #gzip -dc xdk_java.tar | tar xvf -)

The result should be the following files and directories:
/bin - xdk executables and utilities
/lib - directory for libraries
/xdk - top xdk directory
/xdk/demo - directory for demo files
/xdk/doc - directory for documentation
/xdk/admin - direcory for dband config files
/xdk/*html - doc navigation files
/xdk/license.html - copy of license agreement
```

- For Windows NT, choose a directory under which you would like the .\xdk directory and subdirectories to go (for example, \C: on NT), change the directory to \C: then extract the files using the WinZip visual tool.

XDK for JavaBeans Components

After installing the XDK, the directory structure is:

```
-$XDK_HOME
| - bin: executable files and setup script/batch files.
| - lib: library files.
| - xdk
     | - admin (Administration): XSU PL/SQL API setup SQL script and XSL Servlet
           Configuration file (XSQLConfig.xml).
     | - demo: demonstration code
     | - doc: documents including release notes and javadocs.
```

All the packages in XDK for JavaBeans are certified and supported with JDK 1.2 or 1.1.8, so make sure that your CLASSPATH includes all the necessary libraries.

For JDK versions lower that JDK 1.2, you will need to include the JDK library in your CLASSPATH, as well as the Swing library, swingall.jar at "Java Foundation Classes (JFC) / Swing 1.0.3" in page

```
http://java.sun.com/products/archive/index.html
```

The following table lists the libraries of XDK for JavaBeans:

Table 2–5 XDK for JavaBeans Libraries

Component	Library	Notes
XML Parser XSL Processor	xmlparserv2.jar	XML Parser V2 for Java, which includes JAXP 1.1, DOM, SAX, and XSLT APIs.
	xmlmesg.jar	Messages for XML Parser. If you want to use XML Parser with a language other than English, you need to set this jar file in your CLASSPATH.
XML Schema Processor	xschema.jar	XML Schema Processor for Java
XML SQL Utility	xsu12.jar	XML SQL Utility for JDK 1.2 and above
	xsu111.jar	XML SQL Utility for JDK 1.1.8
	oraclexsql.jar	Oracle XSQL Servlet
	xsqlserializers.jar	Oracle XSQL Serializers for FOP / PDF Integration
Class Generator	classgen.jar	Class Generator for Java
TransX Utility	transx.zip	Oracle TransX Utility
JavaBeans	xmlcomp.jar	Oracle JavaBeans Utilities
	xmlcomp2.jar	

In addition, XML SQL Utility, XSQL Servlet and TransX Utility all depend on other components, whose libraries are listed in the following table:

Table 2–6 XDK for JavaBeans: Dependent Libraries

Component	Library	Notes
JDBC	classes12.zip	JDBC for JDK 1.2 and above
	classes111.zip	JDBC for JDK 1.1.8
Globalization	nls_charset12.jar	Globalization support for JDK 1.2 and above
	nls_charset111.jar	Globalization support for JDK 1.1.8
XMLType	xdb_g.jar	XMLType Java APIs. $ORACLE_HOME / rdbms / jlib

Table 2–6 XDK for JavaBeans: Dependent Libraries (Cont.)

Component	Library	Notes
Jdev Runtime	jdev-rt.zip	Java GUI libraries

Setting Up the XDK for JavaBeans Environment

Use this script file provided on UNIX:

```
$XDK_HOME/bin/env.csh
```

For Windows, use this provided batch file:

```
%XDK_HOME/bin/env.bat
```

The following tables list the environment variables needed during XDK setup. Variables that must be customized before running the script or batch file are marked as "Y" in the column "Customize".

Table 2–7 JavaBeans Environment Settings for UNIX

Variable Name	Values	Customize
$JDBCVER	JDBC version. If using JDK 1.2 and above, set to 12.	Y
$JDKVER	JDK version (default is 1.2.2_07), obtained by: `Java -version`	Y
$INSTALL_ROOT	Installation root of XDK, the directory referred to as $XDK_HOME	N
$JAVA_HOME	Directory where the Java SDK, Standard Edition is installed. The path linked to the Java SDK must be modified.	Y
$CLASSPATHJ	${ORACLE_HOME}/jdbc/lib/classes${JDBCVER}.zip: ${ORACLE_HOME}/jdbc/lib/nls_charset${JDBCVER}.jar f you are running the XSU on a system other than where the Oracle RDBMS is installed, you will have to update CLASSPATHJ path with the correct locations of the JDBC library (classes12.jar). The nls_charset12.jar is needed to support certain character sets. Refer to Globalization Support setup with XDK for JavaBeans Note that if you do not have these libraries on your system, these are both available on OTN: (http://otn.oracle.com) which is part of the JDBC driver download.	Y

Table 2–7 JavaBeans Environment Settings for UNIX (Cont.)

Variable Name	Values	Customize
$CLASSPATH	Include the following: .:${CLASSPATHJ}:${INSTALL_ROOT}/lib/xmlparserv2.jar:${INSTALL_ROOT}/lib/xschema.jar: ${INSTALL_ROOT}/lib/xsu${JDBCVER}.jar: ${INSTALL_ROOT}/lib/oraclexsql.jar: ${INSTALL_ROOT}/lib/classgen.jar	N
$PATH	${JAVA_HOME}/bin:${PATH}:${INSTALL_ROOT}/bin	N
$LD_LIBRARY_PATH	For OCI JDBC connections. ${ORACLE_HOME}/lib:${LD_LIBRARY_PATH}	N

For Windows NT see the following table for the settings:

Table 2–8 JavaBeans Environment Settings for Windows NT

Variable Name	Values	Customize
%JDBCVER%	DBC Version. If using JDK 1.2 and above, it is 12. If using JDK 1.1.8, it is 111.	Y
%JDKVER%	JDK version (default is 1.2.2_07), obtained by: `Java -version`	Y
%INSTALL_ROOT%	Installation root of XDK, which is the directory referred to as %XDK_HOME%	N
%JAVA_HOME%	Directory where the Java SDK, Standard Edition, is installed. The path linked to the Java SDK must be modified.	Y
%CLASSPATHJ%	CLASSPATHJ=%ORACLE_HOME%\jdbc\lib\classes%JDBCVER%.zip; %ORACLE_HOME%\jdbc\lib\nls_charset%JDBCVER%.jar	Y
%PATH%	PATH=%JAVA_HOME%\bin;%ORACLE_HOME%\bin;%PATH%;%INSTALL_ROOT%\bin	N
%CLASSPATH%	.;%CLASSPATH%;%INSTALL_ROOT%\lib\xmlparserv2.jar; %INSTALL_ROOT%\lib\xschema.jar; %INSTALL_ROOT%\lib\xsu%JDBCVER%.jar;%INSTALL_ROOT%\lib\oraclexsql.jar;%INSTALL_ROOT%\lib\classgen.jar	N

XDK for JavaBeans with Globalization Support

Here is a summary of the settings that are related to Globalization Support.

- If you use languages other than English, set the `xmlmesg.jar` into your Java `CLASSPATH` to let the parser obtain the correct messages in your language.

- If you use a multibyte character set other than one of the following,
  - UTF-8
  - ISO8859-1
  - JA16SJIS

  then set `nls_charset12.jar` into your Java `CLASSPATH` so that JDBC can convert the character set of the input file to the database character set during loading of XML files.

  See Also: Chapter 10, "XDK JavaBeans" for further discussion of the XDK for JavaBeans components

3

Getting Started with XDKs for C/C++ and PL/SQL

This chapter contains the following sections:

- Installation of XDK for C
- Installation of the XDK for C++
- Installation of XDK for PL/SQL

> **See Also:** Chapter 2, "Getting Started with XDK for Java and JavaBeans"

Installation of XDK for C

XDK for C contains the basic building blocks for reading, manipulating, transforming XML documents.

> **Note:** The XDKs for C and C++ are now bundled together.

Oracle XDK for C consists of the following components:

- XML Parser: supports parsing XML documents with the DOM or SAX interfaces.
- XSL Processor: supports transforming XML documents.
- XML Schema Processor: supports parsing and validating XML files against an XML Schema definition file (default extension `.xsd`).

Getting the XDK for C

If you have installed the Oracle database or iAS (Application Server), you will already have the XDK for C installed.

You can also download the latest versions of XDK for C from OTN.

In order to download the XDK from OTN, follow these steps:

- Use this URL in your browser:
  `http://otn.oracle.com/tech/xml/xdk_c/content.html`
- Click the 'Software' icon at the left-hand side of the page.
- Logon with your OTN username and password (registration is free if you don't already have an account).
- Select the version that you want to download.
- Accept all conditions in the licensing agreement.
- Click the appropriate file
- Extract the files in the distribution:

Refer to "Getting Started with XDK for Java and JavaBeans" on page 2-1 for the details of downloading an XDK (use the XDK for C).

After installing the XDK, the directory structure is:

```
-$XDK_HOME
```

```
| - bin: executable files
| - lib: library files.
|- nlsdata: Globalization Support data files(*.nlb)
| - xdk
     | - demo: demonstration code
     | - doc: documents including release notes.
     | - include: header files.
     | - mesg: message files. (*.msb)
```

Here are all the libraries that come with the UNIX version of XDK for C:

Table 3–1 C for XDK Libraries

Component	Library	Notes
XML Parser XSL Processor	libxml9.a	XML Parser V2 for C, which includes DOM, SAX, and XSLT APIs
XML Schema Processor	libxsd9.a	XML Schema Processor for C

The XDK for C (UNIX) depends on the Oracle CORE and Globalization Support libraries in the following table:

Table 3–2 Dependent Libraries of XDK for C on UNIX

Component	Library	Notes
CORE Library	xmlparser	Oracle CORE library
Globalization Support Library	libnls9.a	Oracle Globalization Support common library
	libunls9.a	Oracle Globalization Support library for Unicode support

UNIX Environment Setup

Check if the environment variable ORA_NLS33 is set to point to the location of the Globalization Support data files.

If you install the Oracle database, you can set it to be:

```
setenv ORA_NLS33 ${ORACLE_HOME}/ocommon/nls/admin/data
```

If no Oracle database is installed, you can set use the Globalization Support data files that come with the XDK release by setting:

```
setenv ORA_NLS33 ${XDK_HOME}/nlsdata
```

Check if the environment variable ORA_XML_MESG is set to point to the absolute path of the `mesg` directory:

If you install the Oracle database, you can set it to be:

```
setenv ORA_NLS33 ${ORACLE_HOME}/xdk/mesg
```

If no Oracle database is installed, you can set it to be the directory of the error message files that come with the XDK release:

```
setenv ORA_NLS33 ${XDK_HOME}/xdk/mesg
```

Currently, all of the message files are in English. The message files for other languages will be provided in a future release.

Now you can use the Makefile to compile and link the demo code.

Windows NT Environment Setup

After installation, the directory structure is:

```
-$XDK_HOME
   | - bin: executable files and dynamic libraries
   | - lib: static library files.
   |- nlsdata: Globalization Support data files (*.nlb)
   | - xdk
        | - demo: demonstration code
        | - doc: documents including release notes.
        | - include: header files.
        | - mesg: message files. (*.msb)
```

These are the Widows NT libraries that come with the XDK for C:

Table 3–3 XDK for C Libraries on NT

Component	Library	Notes
XML Parser XSL Processor	oraxml9.lib oraxml9.dll	XML Parser V2 for C, which includes DOM, SAX, and XSLT APIs
XML Schema Processor	oraxsd9.a oraxsd9.dll	XML Schema Processor for C

The XDK for C (NT) depends on the Oracle CORE and Globalization Support libraries in the following table:

Table 3–4 Dependent Libraries of XDK for C on NT

Component	Library	Notes
CORE Library	oracore9.a	Oracle CORE library
Globalization Support Library	oranls9.a oranls9.dll	Oracle Globalization Support common library
	oraunls9.a oraunls9.dll	Oracle Globalization Support library for Unicode support

Environment for Command Line Usage

Check that the environment variable ORA_NLS33 is set to point to the location of the Globalization Support data files.

If you install the Oracle database, you can set it this way:

```
set ORA_NLS33 =%ORACLE_HOME%\nlsrtl\admin\nlsdata
```

If no Oracle database is installed, you can set use the Globalization Support data files that come with the XDK release:

```
set ORA_NLS33 =%XDK_HOME%\nlsdata
```

You must check if the environment variable ORA_XML_MESG is set to point to the absolute path of the `mesg` directory.

If you install the Oracle database, you can set it to be:

```
set ORA_NLS33 =%ORACLE_HOME%\xdk\mesg
```

If no Oracle database is installed, you can set it to be the directory of the error message files that come with the XDK release:

```
set ORA_NLS33 =%XDK_HOME%\xdk\mesg
```

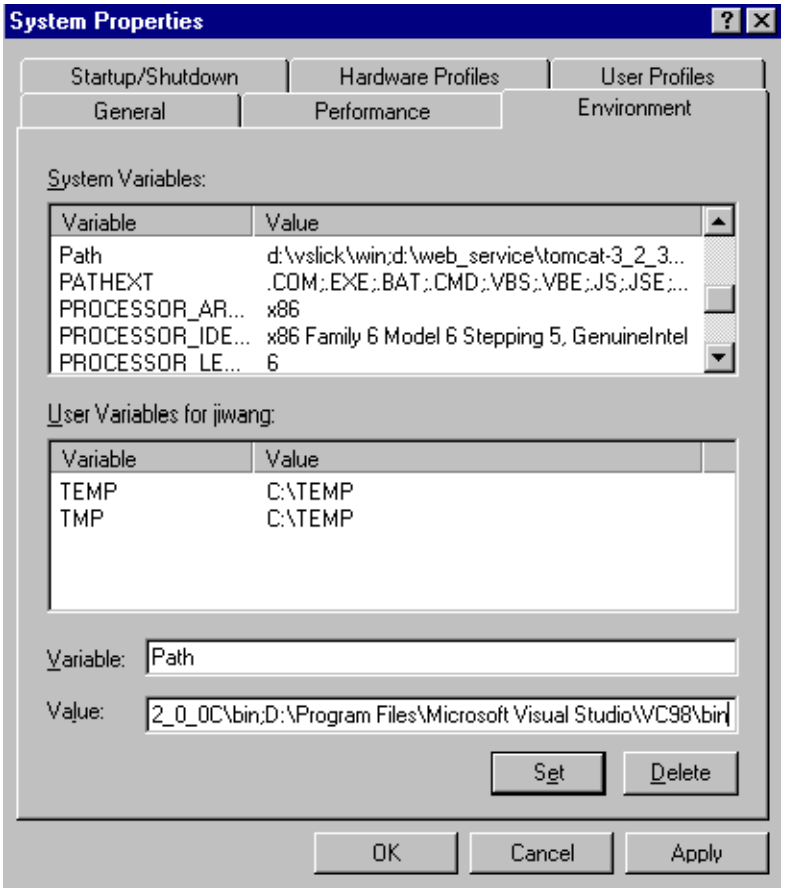
Currently, all of the message files are in English. The message files for other language will be provide in a future release.

Set the path for the `cl` compiler (if you need to compile the code using a Make.bat) in command line environment.

Go to the Start Menu and select Settings > Control Panel. In the pop-up window of Control Panel, select System icon and double click. A window named System Properties will be popped up. Select Environment Tab and input the path of

`cl.exe` to the PATH variable shown in Figure 3–1, "Setting the Path for the cl Compiler in NT".

Figure 3–1 Setting the Path for the `cl` Compiler in NT



You need to update the `Make.bat` by adding the path of the libraries and the header files to the compile and link commands as shown in the following example of a `Make.bat` file:

```
...
: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 -
ID:\Progra~1\Micros~1\VC98\Include %filename%.c
goto :EOF

:LINK
set filename=%1
link %link_dbg% /out:..\..\..\..\bin\%filename%.exe /libpath:%ORACLE_HOME%\lib
/libpath:D:\Progra~1\Micros~1\VC98\lib /libpath:..\..\..\..\lib %filename%.obj
oraxml9.lib oracore9.lib oranls9.lib oraunls9.lib user32.lib kernel32.lib
msvcrt.lib ADVAPI32.lib oldnames.lib winmm.lib
:EOF
```

where:

`D:\Progra~1\Micros~1\VC98\Include:` is the path for header files and `D:\Progra~1\Micros~1\VC98\lib:` is the path for library files.

Using the XDK for C with Visual C++

If you are using Microsoft Visual C++ for your compiler:

Check that the environment variable ORA_NLS33 is set to point to the location of the Globalization Support data files.

If you install the Oracle database, you can set it to be:

```
set ORA_NLS33 =%ORACLE_HOME%\nlsrtl\admin\nlsdata
```

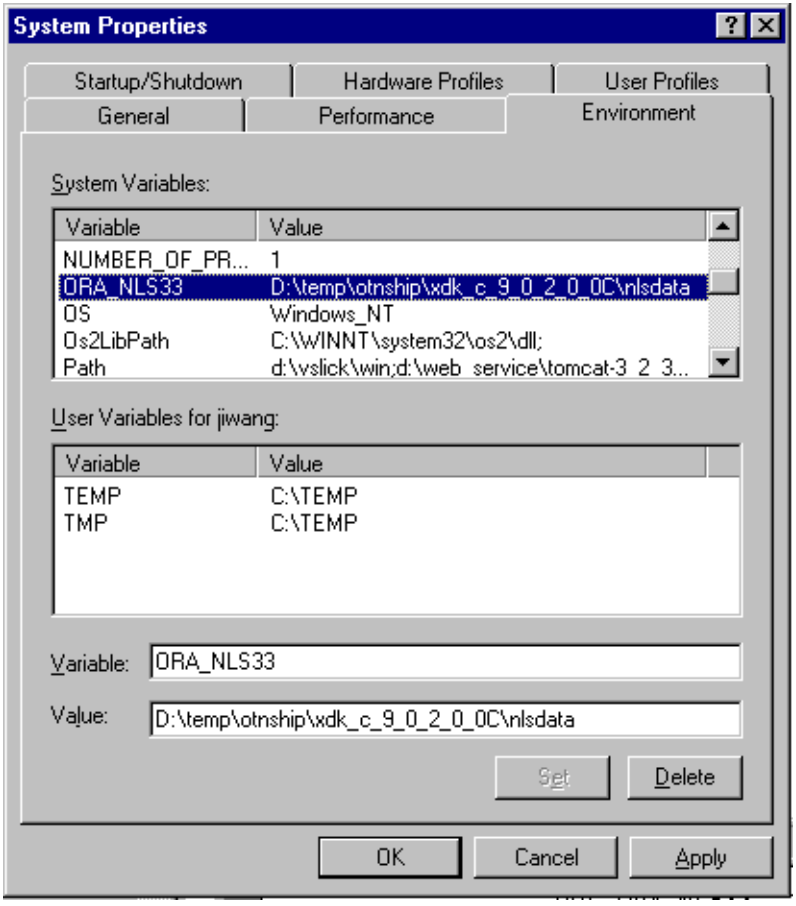
If no Oracle database is installed, you can use the Globalization Support data files that come with the XDK release:

```
set ORA_NLS33 =%XDK_HOME%\nlsdata
```

In order to use Visual C++, you need to employ the system setup for Windows NT to define the environment variable.

Go to Start Menu and select Settings > Control Panel. In the pop up window of Control Panel, select System icon and double click. A window named System Properties will pop up. Select Environment Tab and input ORA_NLS33.

Figure 3–2 Setting Up the ORA_NLS33 Environment Variable



Check that the environment variable ORA_XML_MESG is set to point to the absolute path of the `mesg` directory.

If you install the Oracle database, you can set it to be:

```
set ORA_NLS33 =%ORACLE_HOME%\xdk\mesg
```

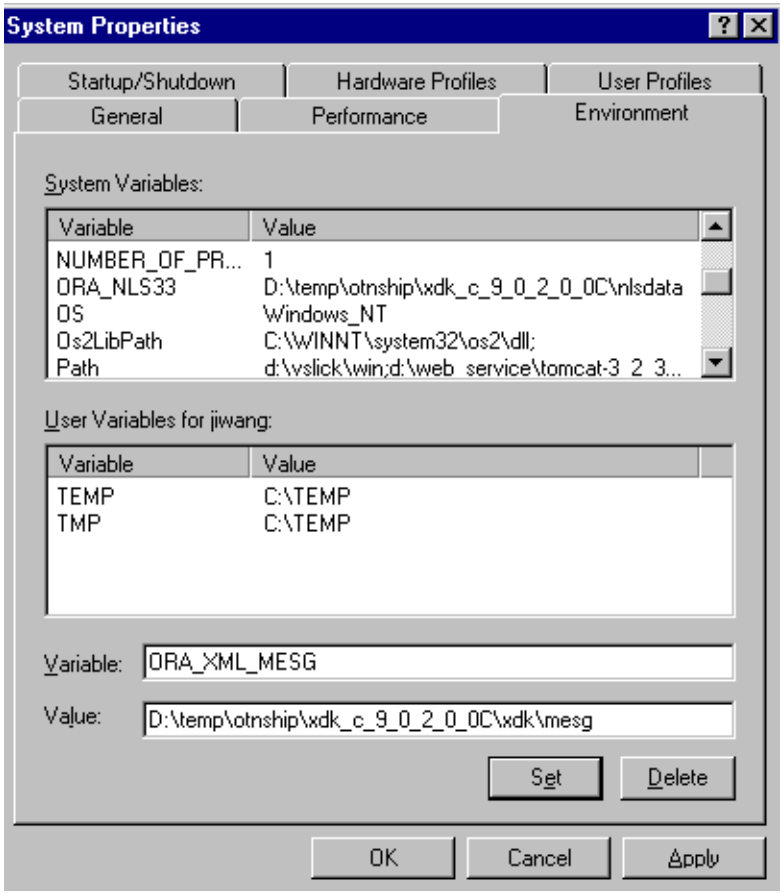
If no Oracle database is installed, you can set it to be the directory of the error message files that come with the XDK release:

```
set ORA_NLS33 =%XDK_HOME%\xdk\mesg
```

In order for Visual C++ to use the environment variable, you need to employ the system setup for windows NT to define the environment variable.

Go to the Start Menu and select Settings > Control Panel. In the pop-up window of Control Panel, select System icon and double click. A window named System Properties will be popped up. Select Environment Tab and input ORA_XML_MESG.

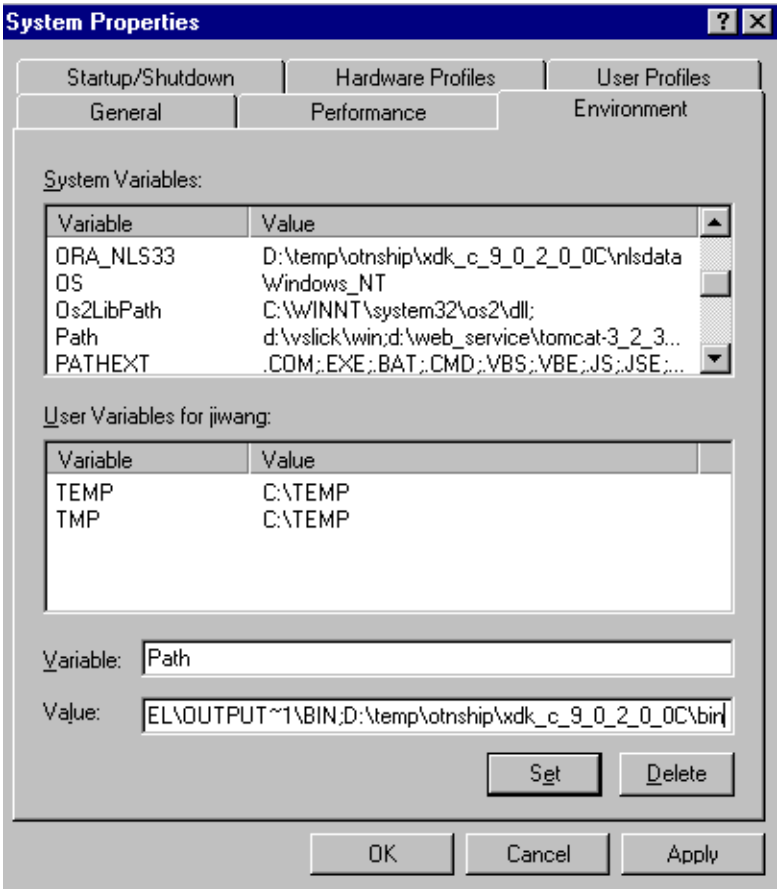
Figure 3–3 Setting Up the ORA_XML_MESG Environment Variable



Currently, all the message files are in English. The message files for other languages will be provided in future releases.

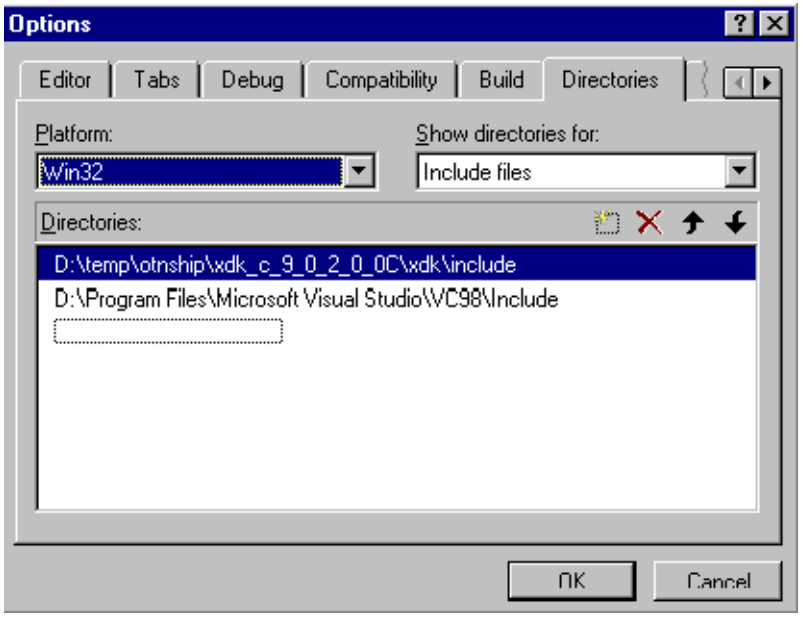
The following figure shows the setup of the PATH for DLLs:

Figure 3–4 Setup of the PATH for DLLs



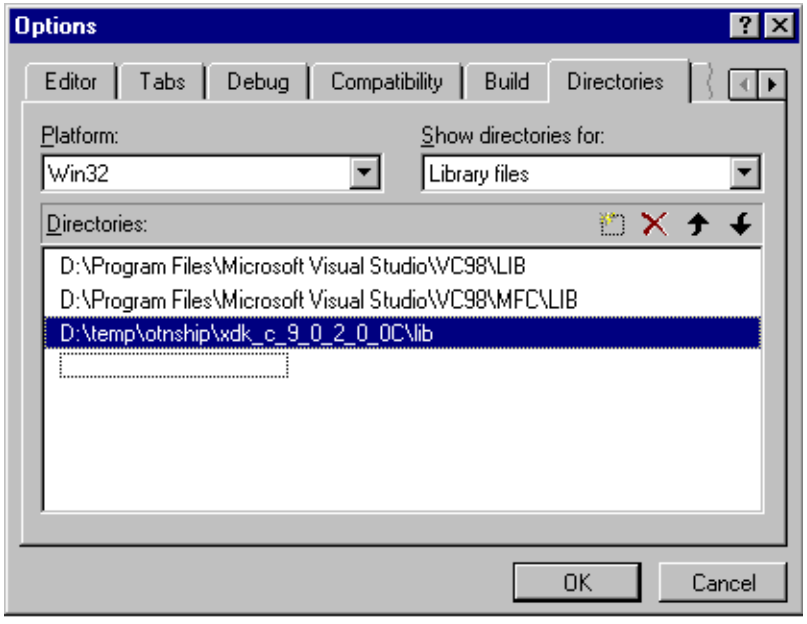
After you open a workspace in Visual C++ and include the `*.c` files for your project, you must set the path for the project. Go to the Tools menu and select Options. A window will pop up. Select the Directory tab and set your include path as shown in the following figure:

Figure 3–5 Setting Your Include Path in Visual C++



Then set your library path as shown in the following figure:

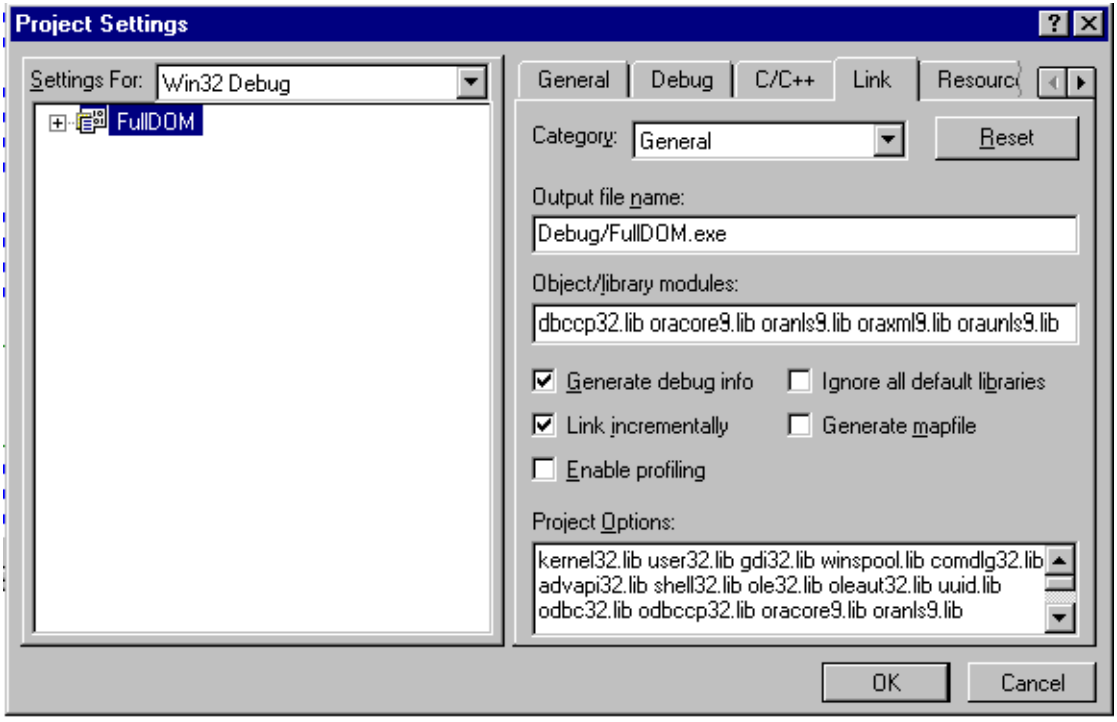
Figure 3–6 Setting Your Static Library Path



After setting the paths for the static libraries in %XDK_HOME\lib, you also need to set the library name in the compiling environment of Visual C++.

Go to the Project menu in the menu bar and select Settings. A window will pop up. Please select the Link tab in the Object/Library Modules field enter the name of XDK for C libraries:

Figure 3–7 Setting Up the Static Libraries in Visual C++ Project



Compile and run the demo programs, and then start using XDK for C.

> **See Also:** Chapter 13, "XML Parser for C" for further discussion of the XDK for C components.

Installation of the XDK for C++

XDK for C++ contains the basic building blocks for reading, manipulating, transforming XML documents.

> **Note:** The XDKs for C and C++ are now bundled together.

Oracle XDK for C consists of the following components:

- XML Parser: supports parsing XML documents with the DOM or SAX interfaces.
- XSL Processor: supports transforming XML documents.
- XML Schema Processor: supports parsing and validating XML files against an XML Schema definition file (default extension `.xsd`).
- Class Generator for C++: generates a set of C++ source files based on an input DTD or XML Schema.

Getting the XDK for C++

If you have installed the Oracle database or iAS (Application Server), you will already have the XDK for C++ installed.

You can also download the latest versions of XDK for C++ from OTN.

In order to download the XDK from OTN, follow these steps:

- Use this URL in your browser:
  `http://otn.oracle.com/tech/xml/xdk_cpp/content.html`
- Click the 'Software' icon at the left-hand side of the page.
- Logon with your OTN username and password (registration is free if you don't already have an account).
- Select the version that you want to download.
- Accept all conditions in the licensing agreement.
- Click the appropriate file
- Extract the files in the distribution:

Refer to "Getting Started with XDK for Java and JavaBeans" on page 2-1 for the details of downloading an XDK (use XDK for C++).

After installing the XDK, the directory structure is:

```
-$XDK_HOME
     | - bin: executable files
     | - lib: library files.
     |- nlsdata: Globalization Support data files(*.nlb)
     | - xdk
          | - demo: demonstration code
          | - doc: documents including release notes.
          | - include: header files.
```

```
| - mesg: message files. (*.msb)
```

The libraries that come with the UNIX version of XDK for C++ are listed in the following table:

Table 3–5 XDK Libraries for C++ (UNIX)

Component	Library	Notes
XML Parser XSL Processor	libxml9.a	XML Parser V2 for C++, which includes DOM, SAX, and XSLT APIs
XML Schema Processor	libxsd9.a	XML Schema Processor for C++
Class Generator	libxmlg.a	Class Generator for C++

The XDK for C++ package depends on the Oracle CORE and Globalization Support libraries, which are listed in the following table:

Table 3–6 Dependent Libraries of XDK for C++ on UNIX

Component	Library	Notes
CORE Library	xmlparser	Oracle CORE library
Globalization Support Library	libnls9.a	Oracle Globalization Support common library
	libunls9.a	Oracle Globalization Support library for Unicode support

Setting the UNIX Environment for C++

Check that the environment variable ORA_NLS33 is set to point to the location of the Globalization Support data files.

If you install the Oracle database, you can set it to be:

```
setenv ORA_NLS33 ${ORACLE_HOME}/ocommon/nls/admin/data
```

If no Oracle database is installed, you can use the Globalization Support data files that come with the XDK release:

```
setenv ORA_NLS33 ${XDK_HOME}/nlsdata
```

Check that the environment variable ORA_XML_MESG is set to point to the absolute path of the `mesg` directory.

If you install the Oracle database, you can set it to be:

```
setenv ORA_NLS33 ${ORACLE_HOME}/xdk/mesg
```

If no Oracle database is installed, you can set it to be the directory of the error message files that comes with the XDK release:

```
setenv ORA_NLS33 ${XDK_HOME}/xdk/mesg
```

Currently, all of the message files are in English. The message files for other languages will be provided in a future release.

You can now use the Makefiles to compile and link the demo code and start developing your program using XDK for C++ on a UNIX platform.

Windows NT Environment Setup

After installation, the directory structure is:

```
-$XDK_HOME
   | - bin: executable files and dynamic libraries
   | - lib: static library files.
   |- nlsdata: Globalization Support data files (*.nlb)
   | - xdk
        | - demo: demonstration code
        | - doc: documents including release notes.
        | - include: header files.
        | - mesg: message files. (*.msb)
```

These are the Widows NT libraries that come with the XDK for C++:

Table 3–7 XDK for C++ Libraries on NT

Component	Library	Notes
XML Parser XSL Processor	oraxml9.lib oraxml9.dll	XML Parser V2 for C++, which includes DOM, SAX, and XSLT APIs.
XML Schema Processor	oraxsd9.a oraxsd9.dll	XML Schema Processor for C++
Class Generator	oraxmlg.a oraxmlg.dll	Class Generator for C++

The XDK for C++ (NT) depends on the Oracle CORE and Globalization Support libraries in the following table:

Table 3–8 Dependent Libraries of XDK for C++ on NT

Component	Library	Notes
CORE Library	oracore9.a oracore9.dll	Oracle CORE library
Globalization Support Library	oranls9.a oranls9.dll	Oracle Globalization Support common library
	oraunls9.a oraunls9.dll	Oracle Globalization Support library for Unicode support

Command Line Usage

Check that the environment variable ORA_NLS33 is set to point to the location of the Globalization Support data files.

If you install the Oracle database:

```
set ORA_NLS33 =%ORACLE_HOME%\nlsrtl\admin\nlsdata
```

If no Oracle database is installed, you can use the Globalization Support data files that come with the XDK release:

```
set ORA_NLS33 =%XDK_HOME%\nlsdata
```

Check that the environment variable ORA_XML_MESG is set to point to the absolute path of the `mesg` directory.

If you install the Oracle database, you can set it to be:

```
set ORA_NLS33 =%ORACLE_HOME%\xdk\mesg
```

If no Oracle database is installed, you can set it to be the directory of the error message files, which comes with the XDK release:

```
set ORA_NLS33 =%XDK_HOME%\xdk\mesg
```

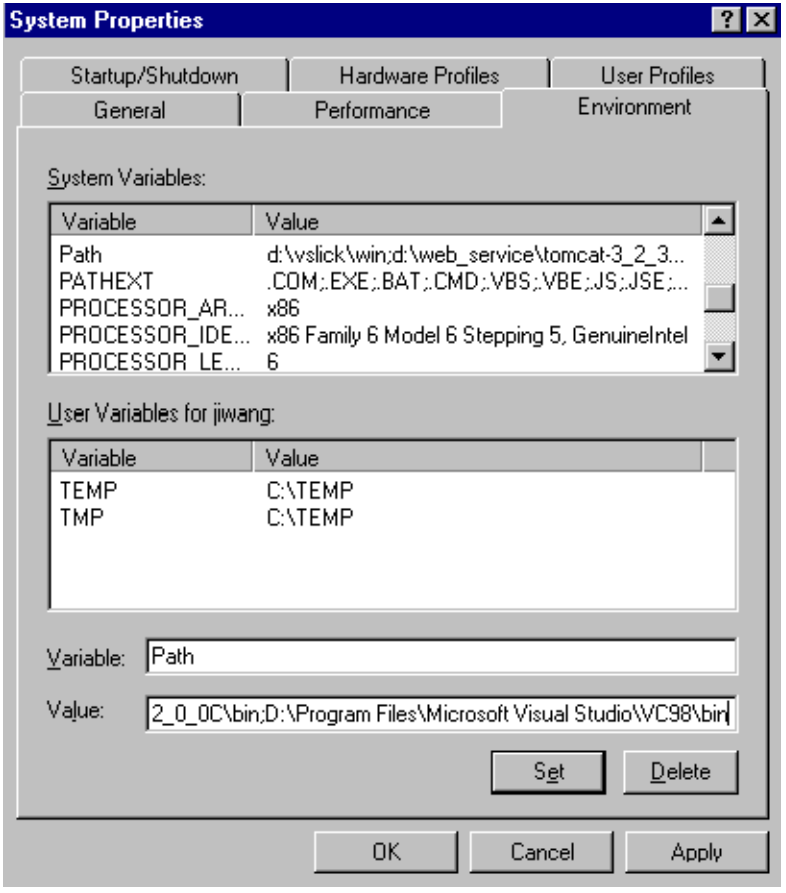
Currently, all of the message files are in English. The message files for other languages will be provided in a future release.

Set the path for `cl` compiler, if you need to compile the code using `make.bat` in a command line.

Go to the Start Menu and select Settings > Control Panel. In the pop up window of Control Panel, select System icon and double click. A window named System

Properties will pop up. Select Environment Tab and input the path of `cl.exe` to the PATH variable shown in Figure 3–8, "Setting the PATH for the cl Compiler".

Figure 3–8 Setting the PATH for the `cl` Compiler



You must update the file `Make.bat` by adding the path of the libraries and header files to the compile and link commands:

```
...
: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 -
ID:\Progra~1\Micros~1\VC98\Include %filename%.c
goto :EOF

:LINK
set filename=%1
link %link_dbg% /out:..\..\..\..\bin\%filename%.exe /libpath:%ORACLE_HOME%\lib
/libpath:D:\Progra~1\Micros~1\VC98\lib /libpath:..\..\..\..\lib %filename%.obj
oraxml9.lib oracore9.lib oranls9.lib oraunls9.lib user32.lib kernel32.lib
msvcrt.lib ADVAPI32.lib oldnames.lib winmm.lib

:EOF
...
```

where

`D:\Progra~1\Micros~1\VC98\Include:` is the path for header files and `D:\Progra~1\Micros~1\VC98\lib:` is the path for library files.

Now you can start developing with XDK for C++.

Using XDK for C++ with Visual C ++

Check that the environment variable ORA_NLS33 is set to point to the location of the Globalization Support data files.

If you install the Oracle database, you can set it to be

```
set ORA_NLS33 =%ORACLE_HOME%\nlsrtl\admin\nlsdata
```

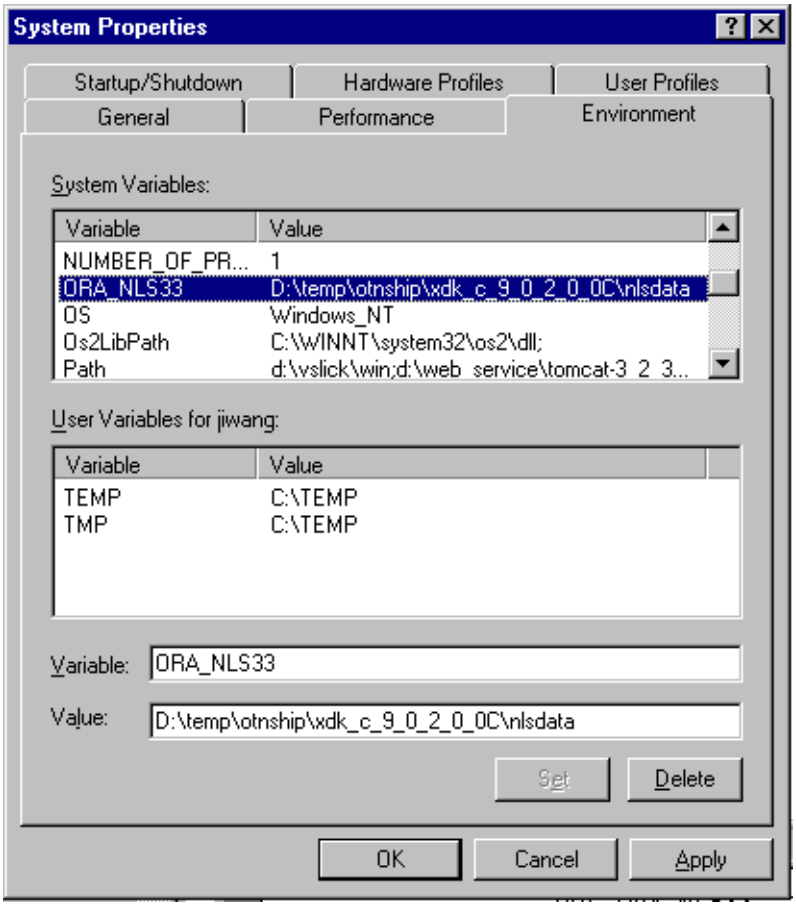
If no Oracle database is installed, you can use the Globalization Support data files that come with the XDK release:

```
set ORA_NLS33 =%XDK_HOME%\nlsdata
```

In order for Visual C++ to know the environment variable, you need to use the system setup for windows NT to define the environment variable.

Go to Start Menu and select Settings > Control Panel. In the pop-up window of Control Panel, select System icon and double click. A window named System Properties will be popped up. Select Environment Tab and input ORA_NLS33.

Figure 3–9 Setting Up the ORA_NLS33 Environment Variable



Check that the environment variable ORA_XML_MESG is set to point to the absolute path of the `mesg` directory.

If you install the Oracle database, you can set it:

```
set ORA_NLS33 =%ORACLE_HOME%\xdk\mesg
```

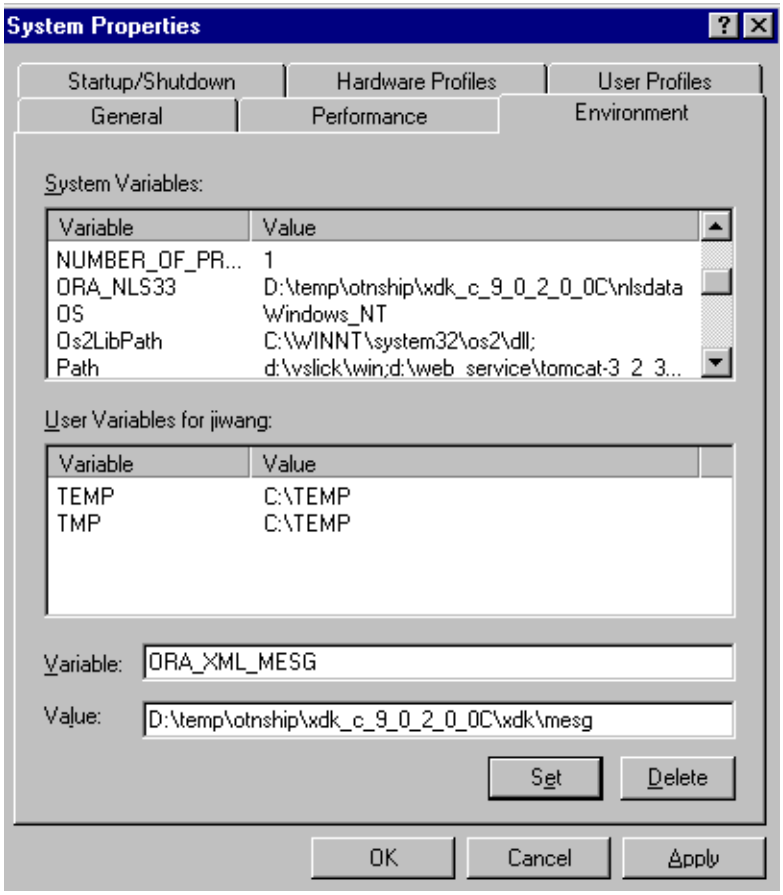
If no Oracle database is installed, you can set it to be the directory of the error message files that comes with the XDK release:

```
set ORA_NLS33 =%XDK_HOME%\xdk\mesg
```

In order for Visual C++ to employ the environment variable, you need to use the system setup for Windows NT to define the environment variable.

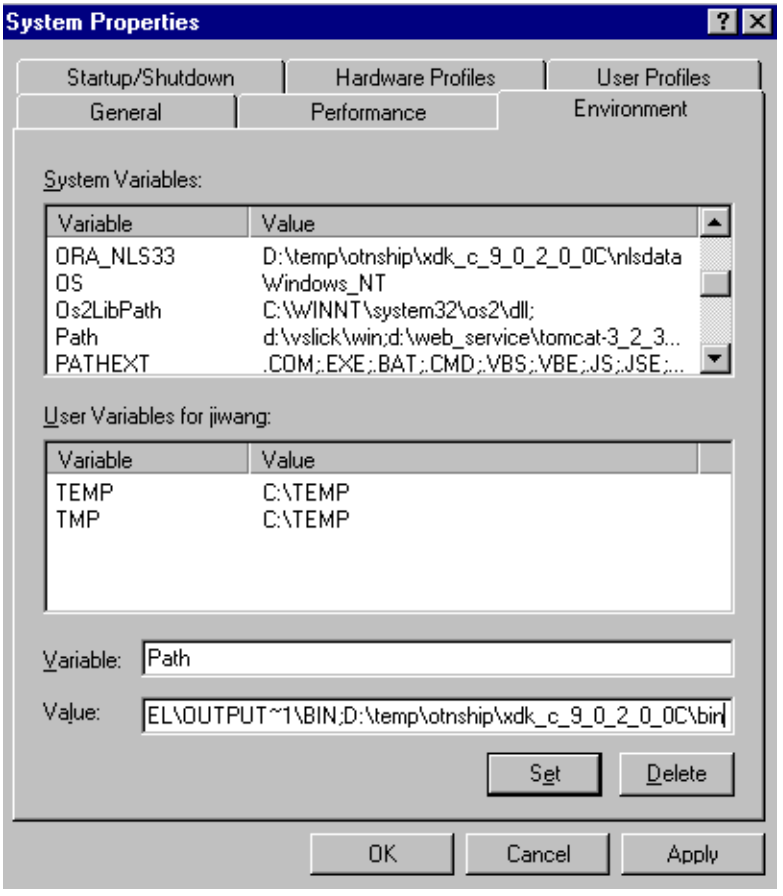
Go to the Start Menu and select Settings > Control Panel. In the pop-up window of Control Panel, select System icon and double click. A window named System Properties will pop up. Select Environment Tab and input the ORA_XML_MESG.

Figure 3–10 Setting Up ORA_XML_MESG Environment Variable



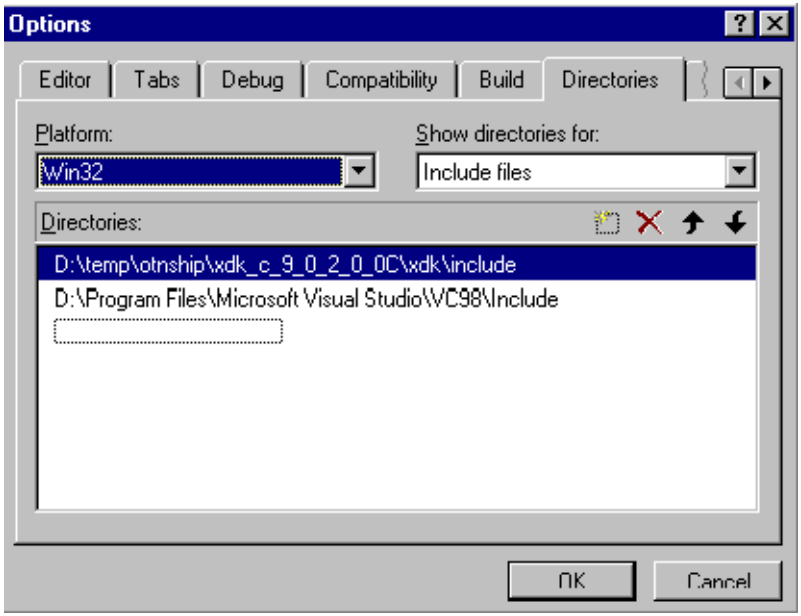
Currently, all of the message files are in English. The message files for other languages will be provided in a future release.

Figure 3–11 Setup of the PATH for DLLs



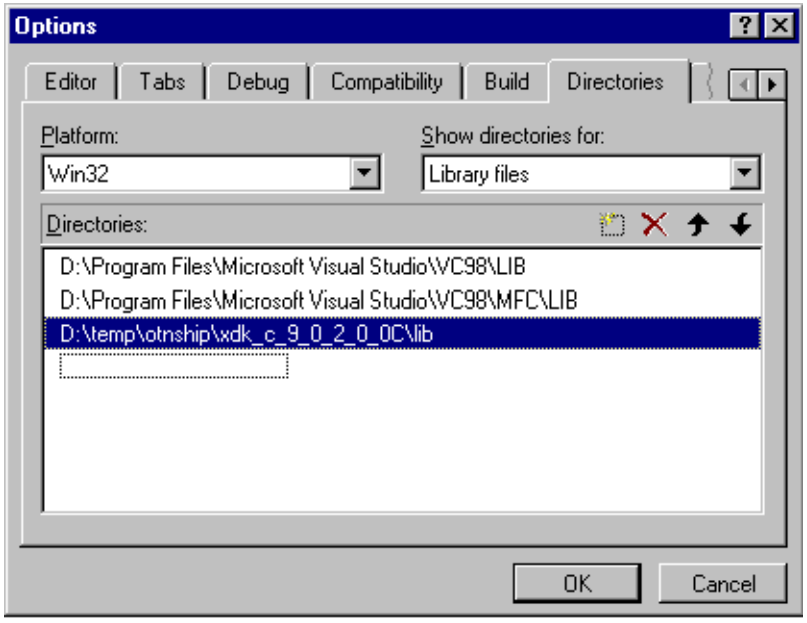
After you open a workspace in Visual C++ and include the `*.c` files for your project, you must set the path for the project. Go to the Tools menu and select Options. A window will pop up. Select the Directory tab and set your include path as shown in the following figure:

Figure 3–12 Setting Your Include Path in Visual C++



Then set your library path as shown in the following figure:

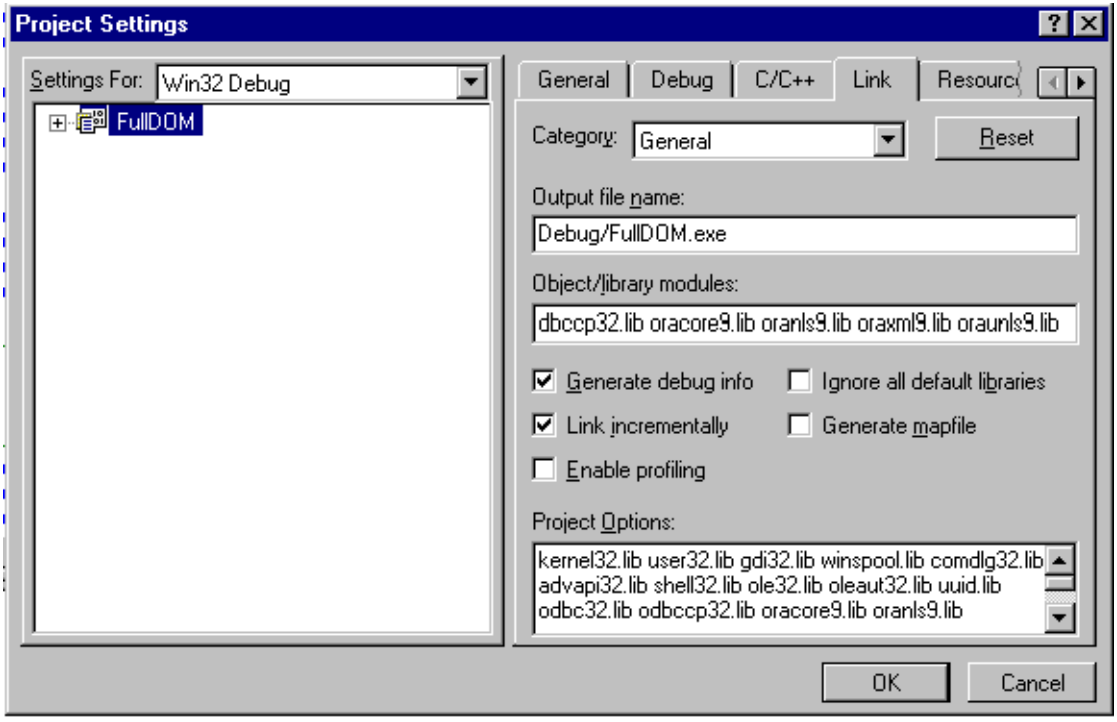
Figure 3–13 Setting Your Static Library Path



After setting the paths for the static libraries in %XDK_HOME\lib, you also need to set the library name in the compiling environment of Visual C++.

Go to the Project menu in the menu bar and select Settings. A window will pop up. Please select the Link tab in the Object/Library Modules field enter the name of XDK for C++ libraries:

Figure 3–14 Setting Up the Static Libraries in Visual C++ Project



You can now compile and run the demo programs, and start using XDK for C++.

See Also: Chapter 16, "XML Parser for C++" for further discussion of the XDK for C++ components

Installation of XDK for PL/SQL

XDK for PL/SQL contains the basic building blocks for reading, manipulating, and transforming XML documents. Oracle XDK for PL/SQL consists of the following components:

- XML Parser: supports parsing XML documents with the DOM interfaces.
- XSL Processor: supports transforming XML documents.
- XML SQL Utility: generates an XML Document from SQL queries and inserts an XML document into the database.

Setting the Environment for XDK for PL/SQL

If you have installed the Oracle database or iAS (Application Server), you will already have the XDK for PL/SQL installed.

You can also download the latest versions of XDK for PL/SQL from OTN.

In order to download the XDK from OTN, follow these steps:

- Use this URL in your browser:
  `http://otn.oracle.com/tech/xml/xdk_plsql/content.html`
- Click the 'Software' icon at the left-hand side of the page.
- Logon with your OTN username and password (registration is free if you don't already have an account).
- Select the version that you want to download.
- Accept all conditions in the licensing agreement.
- Click the appropriate file
- Extract the files in the distribution:

Refer to "Getting Started with XDK for Java and JavaBeans" on page 2-1 for the details of downloading an XDK (using the XDK for PL/SQL).

After installing the XDK, the directory structure is:

```
-$XDK_HOME
    | - bin: executable files and setup script/batch files.
    | - lib: library files.
    | - xdk:
         | - admin: (Administration): XSU PL/SQL API setup SQL script
               and XSL Servlet Configuration file(XSQLConfig.xml).
         | - demo: demonstration code
         | - doc: documents including release notes and javadocs.
```

The following table lists all the Java libraries that come with XDK for PL/SQL:

Table 3–9 XDK Libraries for PL/SQL

Component	Library	Notes
XML Parser XSL Processor	xmlparserv2.jar	XML Parser V2 for Java, which includes JAXP 1.1, DOM, SAX and XSLT APIs.
	xmlmesg.jar	Message files for XML Parser. If you want to use XML Parser with a language other than English, you need to set this JAR file in your CLASSPATH.
XML Schema Processor	xschema.jar	XML Schema Processor for Java.
XML SQL Utility	xsu12.jar	XML SQL Utility for JDK 1.2 and above.
	xsu111.jar	XML SQL Utility for JDK 1.1.8.
XML PL/SQL Package	xmlplsql.jar	XML PL/SQL package.

The PL/SQL packages provided are listed in the following table:

Table 3–10 XDK Packages for PL/SQL

PL/SQL Library	Package Name	Notes
XML Parser	xmlparser	XML Parser.
	xmldom	DOM API for XML.
XSL Processor	xslprocessor	XML Schema Processor for PL/SQL.
XML SQL Utility	DBMS_XMLQuery	XML SQL Utility PL/SQL package reflects the functions in the Java classes – OracleXMLQuery. It is used to generate XML from SQL queries.
	DBMS_XMLSave	XML SQL Utility PL/SQL package reflects the functions in the Java classes – OracleXMLSave. It is used to store XML into the database.

Installing XDK for PL/SQL into the Database

Before installing the XDK for PL/SQL packages into the database, you need to check the status of the packages and the related Java libraries.

Checking PL/SQL Package Status

You can use the following command to check if any of the PL/SQL packages is in your current database schema:

```
SQL*PLUS> desc package_name
```

For example:

```
SQL*PLUS> desc xmldom
```

If you see the content of the package, then the package is available to be used in your schema and you can skip all of the rest of the installation steps.

If you see the following error messages:

```
SQL> desc xmldom
ERROR:
OrA-04043: object "SYS"."XMLDOM" does not exists.
```

it means that the XDK for PL/SQL packages have not been defined in your database schema. You need to do the status checking for the related Java libraries.

Checking Java Libraries Status

The libraries, including `xmlparserv2.jar`, `xmlplsql.jar` and `xsu12.jar` (or `xsu111.jar`), are required to be loaded to the database. You can use SQL commands to check the status of a specific library by the classes that the library contains.

For example, to check the status of xmlparserv2.jar, you can check the classes within oracle.xml.parser.v2.DOMParser class by using the following SQL statement:

```
SELECT SUBSTR(dbms_java.longname(object_name),1,35) AS class, status
     FROM   all_objects
     WHERE  object_type = 'JAVA CLASS'
     AND object_name = dbms_java.shortname('oracle/xml/parser/v2/DOMParser');
```

If you see the result:

```
CLASS                                               STATUS
-----------------------------------------------------------
oracle/xml/parser/v2/DOMParser                       INVALID
```

then try the command:

```
ALTER JAVA CLASS _oracle/xml/parser/v2/DOMParser Resolve
```

If the verification procedure produces the SQL*Plus message “no rows selected”, you need to use the XDKLOAD utility in "Loading XDK for PL/SQL" on page 3-29.

If you see the preceding result, but the status is VALID, that means the Oracle XML Parser for Java is already installed and ready to be used. If all of the Java libraries have already been loaded into the database, then you can run the SQL scripts to define the PL/SQL packages.

For SYS users who would like to create public synonyms in addition to the packages:

For XML Parser and PL/SQL:

```
$XDK_HOME/xdk/admin/xmlpkg.sql
$XDK_HOME/xdk/admin/xmlsyn.sql
```

For XSU:

```
$XDK_HOME/xdk/admin/xsupkg.sql
$XDK_HOME/xdk/admin/xsusyn.sql
```

For all other users:

For XML Parser and PL/SQL:

```
$XDK_HOME/xdk/admin/xmlpkg.sql
```

For XSU:

```
$XDK_HOME/xdk/admin/xsupkg.sql
```

If any single library is not valid you can load the package by directly using the load Java utility:

```
loadjava -resolve -verbose -user  xdktemp/xdktemp xmlparserv2.jar
```

Loading XDK for PL/SQL

Before using LOADJAVA utility to load the Java libraries into the database schema, you need to get the Java VM properly installed. You have to run the INITJVM.SQL and INITDBJ.SQL scripts to initialize the Java environment before running the LOADJAVA utility. Usually these are in the $ORACLE_HOME/javavm/install subdirectory of your Oracle Home directory.

Using xdkload

To load the XDK for PL/SQL packages into the database schema, you can use the script or batch files provided by XDK.

UNIX:

```
$XDK_HOME/bin/xdkload
```

Windows:

```
$XDK_HOME/bin/xdkload.bat
```

The `xdkload` command syntax is:

```
xdkload -u username/password [-s] [-noverify] [-dbver]

-s         Creates public synonyms for the loaded java APIs; this can be invoked
           only if the target user has dba privileges.
-noverify  Use this if you are loading into an older version of the db and ar
           running into an error about missing method (for example, if
           you are loading xsu version 9.0.1.0.0 into Oracle 8.1.7).
-dbver     Used to specify the version of the database into which you are
           loading XDK. This is a must if Oracle older than the version of
           the XDK). This option also sets the -noverify option.
```

For example:

```
xdkload -u "system/manager" -s -dbver "816"
```

This example uses xdkload to load the XDK for PL/SQL packages to system user.

Before using xdkload, you need to check if any of the libraries including `xmlparserv2.jar`, `xmlxsql.jar` and `xsu12.jar` (`xsu111.jar`) is already loaded to the database. If so, you need to drop them before using `xdkload`:

```
dropjava -verbose -user xdktemp/xdktemp  xmlparserv2.jar xschema.jar
```

Moreover, you need to set up the environment variables by using the script or batch file XDK provides:

UNIX:

```
$XDK_HOME/bin/env.csh
```

Windows:

```
$XDK_HOME/bin/env.bat
```

You can refer to "Getting Started with XDK for Java and JavaBeans" on page 2-1 for the detailed information of this environment setup.

After running the `xdkload` script or batch file, if the target user name used to run `xdkload` has DBA privileges, then the XDK for PL/SQL package will be available to all the users and the public synonyms for the PL/SQL packages are also created. Otherwise, the XDK for PL/SQL packages will be available only to the target user.

> **See Also:** Chapter 20, "XML Parser for PL/SQL" or further discussion of the XDK for PL/SQL components

Part II

XDK for Java

These chapters describe how to access and use the XDK for Java:

- Chapter 4, "XML Parser for Java"
- Chapter 5, "XSLT Processor for Java"
- Chapter 6, "XML Schema Processor for Java"
- Chapter 7, "XML Class Generator for Java"
- Chapter 8, "XML SQL Utility (XSU)"
- Chapter 9, "XSQL Pages Publishing Framework"
- Chapter 10, "XDK JavaBeans"
- Chapter 11, "Using XDK and SOAP"
- Chapter 12, "Oracle TransX Utility"

Note:

- XML-SQL Utility (XSU) is also considered part of the XDK for Java (and the XDK for PL/SQL). In this manual, XSU is described in Chapter 8, "XML SQL Utility (XSU)".
- XSQL Servlet is considered part of XDK for Java. In this manual XSQL Servlet is described in Chapter 9, "XSQL Pages Publishing Framework"

4

XML Parser for Java

This chapter contains the following sections:

XML Parser for Java: Features

Oracle provides a set of XML parsers for Java, C, C++, and PL/SQL. Each of these parsers is a standalone XML component that parses an XML document (or a standalone DTD or XML Schema) so that it can be processed by an application. This chapter discusses the parser for Java only. The other language versions are discussed in later chapters.

Library and command-line versions are provided supporting the following standards and features:

- *XML*. W3C XML 1.0 Recommendation
- DOM. Integrated DOM (Document Object Model) API, compliant with:
  - W3C DOM 1.0 Recommendation
  - W3C DOM 2.0 CORE Recommendation and Mutation Event.
  - W3C DOM 2.0 Traversal Recommendation, including Treewalker, Node Iterator, and Node Filter.
  - DOM level XML compression.

  These APIs permit applications to access and manipulate an XML document as a tree structure in memory. This interface is used by such applications as editors.
- *SAX*. Integrated SAX (Simple API for XML) API, compliant with the SAX 2.0 recommendation and with SAX2-ext. These APIs permit an application to process XML documents using an event-driven model.
- W3C Proposed Recommendation for XML Namespaces 1.0 thereby avoiding name collision, increasing reusability and easing application integration. Supports Oracle XML Schema Processor.

  See Also: .
  `http://www.w3.org/TR/1999/REC-xml-names-19990114/`

- *XSLT*. XSLT Processor for Java includes the following features:
  - Integrated support for W3C XSLT 1.1 Working Draft
  - Provides new APIs to get XSL Transformation as SAX Output
- XML Schema Processor. Supports XML Schema Processor that parses and validates XML files against an XML Schema Definition file (.xsd). It includes the following features:

  - Built on the XML Parser for Java v2
  - Supports the three parts of the XML Schema Working Draft
    * Part 0: Primer XML Schema
    * Part 1: Structures XML Schema
    * Part 2: Datatypes
- Runs on Oracle9*i* and Oracle9*i* Application Server

Additional features include:

- Built-in error recovery until fatal error
- Support for JAXP 1.1.

The parsers are available on all Oracle platforms.

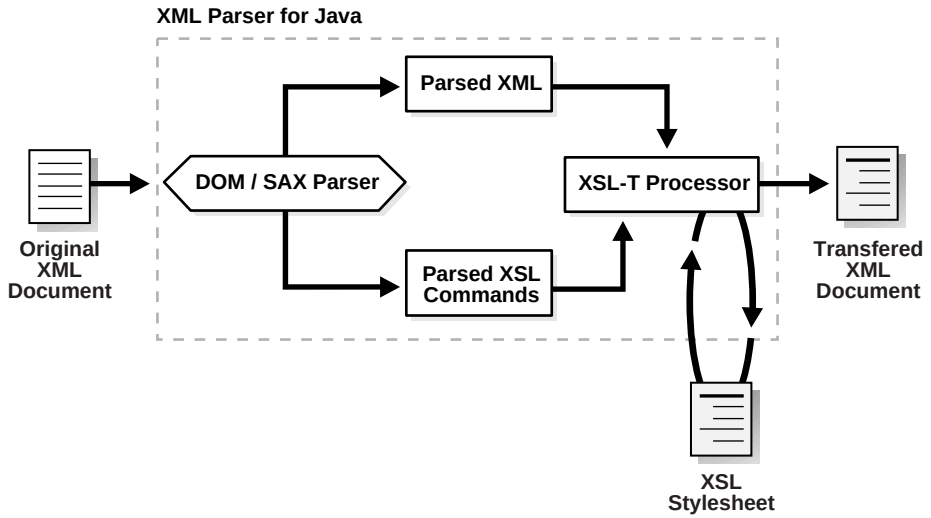
Figure 4–1 shows an XML document inputting XML Parser for Java. The DOM or SAX parser interface parses the XML document. The parsed XML is then transferred to the application for further processing.

If a stylesheet is used, the DOM or SAX interface also parses and outputs the XSL commands. These are sent together with the parsed XML to the XSLT Processor where the selected stylesheet is applied and the transformed (new) XML document is then output.

See Also:

- Appendix A, "XDK for Java: Specifications and Quick References"
- Chapter 5, "XSLT Processor for Java"
- Chapter 6, "XML Schema Processor for Java"

Figure 4–1 Oracle XML Parser



DOM and SAX APIs are explained in "DOM and SAX APIs".

The classes and methods used to parse an XML document are illustrated in the following diagrams:

- Figure 4–4, "XML Parser for Java: DOMParser()"
- Figure 4–5, "Using SAXParser() Class"

The classes and methods used by the XSLT Processor to apply stylesheets are illustrated in the following diagram:

- Figure 5–1, "Using XSL Processor for Java"

XSL Transformation (XSLT) Processor

The V2 versions of the XML Parsers include an integrated XSL Transformation (XSLT) Processor for transforming XML data using XSL stylesheets. Using the XSLT processor, you can transform XML documents from XML to XML, XML to HTML, or to virtually any other text-based format. See Figure 4–1.

> **See Also:** Chapter 5, "XSLT Processor for Java" for complete details.

Namespace Support

The Java XML parser also supports XML Namespaces. Namespaces are a mechanism to resolve or avoid name collisions between element types (tags) or attributes in XML documents.

This mechanism provides "universal" namespace element types and attribute names whose scope extends beyond this manual.

Such tags are qualified by uniform resource identifiers (URIs), such as:

```
<oracle:EMP xmlns:oracle="http://www.oracle.com/xml"/>
```

For example, namespaces can be used to identify an Oracle <EMP> data element as distinct from another company's definition of an <EMP> data element.

This enables an application to more easily identify elements and attributes it is designed to process. The Java parser supports namespaces by being able to recognize and parse universal element types and attribute names, as well as unqualified "local" element types and attribute names.

See Also:

- Chapter 6, "XML Schema Processor for Java"
- *Oracle9i XML API Reference - XDK and Oracle XML DB*

Oracle XML Parsers Validation Modes

The Java parser can parse XML in validating or non-validating modes.

- **Non-Validating Mode**. The parser verifies that the XML is well-formed and parses the data into a tree of objects that can be manipulated by the DOM API.
- **DTD Validating Mode.** The parser verifies that the XML is well-formed and validates the XML data against the DTD (if any).
- **Partial Validation Mode.** Partial validation validates an input XML document according to the DTD if a DTD or XMLS Schema is present else it will be in Non-Validating mode.
- **Schema Validation Mode.** The XML Document is validated according to the XML Schema specified for the document.
- **Auto Validation Mode.** In this mode the parser does its best to validate with whatever is available. If DTD is available, it is set to DTD_VALIDATION, if Schema is present then it is set to SCHEMA_VALIDATION. If none is available, it is set to NON_VALIDATING mode.

Validation involves checking whether or not the attribute names and element tags are legal, whether nested elements belong where they are, and so on.

See Also: *Oracle9i XML API Reference - XDK and Oracle XML DB*

Parsers Access XML Document's Content and Structure

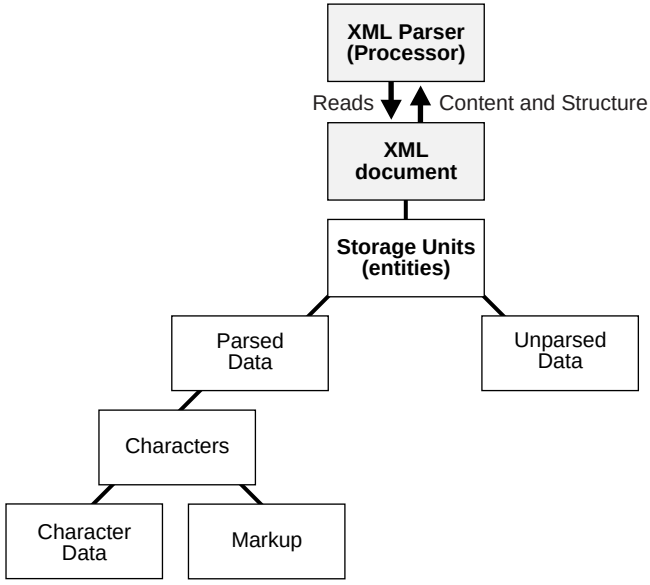
XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup.

Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constraints on the storage layout and logical structure.

A software module called an XML processor is used to read XML documents and provide access to their content and structure. It is assumed that an XML processor is doing its work on behalf of another module, called the application.

This parsing process is illustrated in Figure 4–2.

Figure 4–2 XML Parsing Process



DOM and SAX APIs

XML APIs generally fall into the following two categories:

- Event-based
- Tree-based

See Figure 4–3. Consider the following simple XML document:

```
<?xml version="1.0"?>
  <EMPLIST>
    <EMP>
     <ENAME>MARY</ENAME>
    </EMP>
    <EMP>
     <ENAME>SCOTT</ENAME>
    </EMP>
  </EMPLIST>
```

DOM: Tree-Based API

A tree-based API (such as DOM) builds an in-memory tree representation of the XML document. It provides classes and methods for an application to navigate and process the tree.

In general, the DOM interface is most useful for structural manipulations of the XML tree, such as reordering elements, adding or deleting elements and attributes, renaming elements, and so on. For example, for the XML document preceding, the DOM creates an in-memory tree structure as shown in Figure 4–3.

SAX: Event-Based API

An event-based API (such as SAX) uses calls to report parsing events to the application. The application deals with these events through customized event handlers. Events include the start and end of elements and characters.

Unlike tree-based APIs, event-based APIs usually do not build in-memory tree representations of the XML documents. Therefore, in general, SAX is useful for applications that do not need to manipulate the XML tree, such as search operations, among others.

The preceding XML document becomes a series of linear events as shown in Figure 4–3.

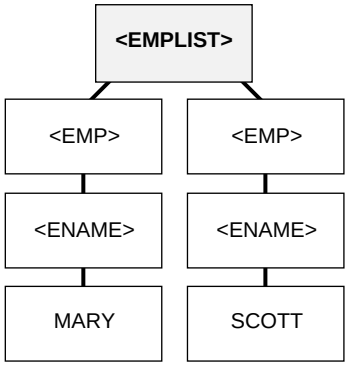
Figure 4–3 Comparing DOM (Tree-Based) and SAX (Event-Based) APIs

[

XML Document

```
<?XML Version = "1.0"?>
  <EMPLIST>
    <EMP>
      <ENAME>MARY</ENAME>
    </EMP>
    <EMP>
      <ENAME>SCOTT</ENAME>
    </EMP>
  </EMPLIST>
```

The DOM interface creates a TREE structure based on the XML Document



Useful for applications that include changes eg. reordering, adding, or deleting elements.

The SAX interface creates a series of linear events based on the XML document

```
start document

start element: EMPLIST
start element: EMP
start element: ENAME
characters: MARY
end element: EMP

start element: EMP
start element: ENAME
characters: SCOTT
end element: EMP

end element: EMPLIST
end document
```

Useful for applications such as search and retrieval that do not change the "XML tree".

Guidelines for Using DOM and SAX APIs

Here are some guidelines for using the DOM and SAX APIs:

DOM:

- Use the DOM API when you need to use random access.
- DOM consumes more memory.
- Use DOM when you are performing transformations.
- Use DOM when you want to have tree iterations and need to walk through the entire document tree.
- When using the DOM interface, try to use more attributes over elements in your XML, to reduce the pipe size.

SAX:

Use the SAX API when your data is mostly streaming data.

XML Compressor

This release supports binary compression of XML documents. The compression is based on tokenizing the XML tags. The assumption is that any XML document has a repeated number of tags and so tokenizing these tags will give considerable amount of compression. Therefore the compression achieved depends on the type of input document; the larger the tags and the lesser the text content, then the better the compression.

The goal of compression is to reduce the size of the XML document without loosing the structural and hierarchical information of the DOM tree. The compressed stream contains all the "useful" information to create the DOM tree back from the binary format. The compressed stream can also be generated from the SAX events. The binary stream generated from DOM and SAX are compatible. The compressed stream generated from SAX could be used to generate the DOM tree and vice versa.

Sample programs to illustrate the compression feature is included in demos.

Oracle XML Parser can also compress XML documents. Using the compression feature, an in-memory DOM tree or the SAX events generated from an XML document can be compressed to generate a binary compressed output.

The compressed stream generated from DOM and SAX are compatible, that is, the compressed stream generated from SAX could be used to generate the DOM tree and vice versa. The compression is based on tokenizing the XML tags. This is based on the assumption that XML files typically have repeated tags and tokenizing the tags compresses the data. The compression depends on the type of input XML document: the larger the number of tags, the less the text content, and the better the compression.

As with XML documents in general, you can store the *compressed* XML data output as a BLOB (Binary Large Object) in the database.

XML Serialization/Compression

An XML document is compressed into a binary stream by means of the serialization of an in-memory DOM tree. When a large XML document is parsed and a DOM tree is created in memory corresponding to it, it may be difficult to satisfy memory requirements and this could affect performance. The XML document is compressed into a byte stream and stored in an in-memory DOM tree. This can be expanded at a later time into a DOM tree without performing validation on the XML data stored in the compressed stream.

The compressed stream can be treated as a serialized stream, but note that the information in the stream is more controlled and managed, compared to the compression implemented by Java's default serialization.

In this release, there are two kinds of XML compressed streams:

- *SAX based Compression*: The compressed stream is generated when an XML file is parsed using a SAX Parser. SAX events generated by the SAX Parser are handled by the SAX Compression utility. It handles the SAX events to generate a compressed binary stream. When the binary stream is read back, the SAX events are generated.
- *DOM based compression:* The in-memory DOM tree, corresponding to a parsed XML document, is serialized, and a compressed XML output stream is generated. This serialized stream when read back regenerates the DOM tree.

The compressed stream is generated using SAX events and that generated using DOM serialization are compatible. You can use the compressed stream generated by SAX events to create a DOM tree and vice versa. The compression algorithm used is based on tokenizing the XML tag's. The assumption is that any XML file has repeated number of tags and therefore tokenizing the tags will give considerable compression.

Running the XML Parser for Java Samples

The directory `demo/java/parser` contains some sample XML applications to show how to use the Oracle XML parser.

The following are the sample Java files in the sub-directories:

- XSLSample - A sample application using XSL APIs.
- DOMSample - A sample application using DOM APIs.
- DOMNamespace - A sample application using Namespace extensions to DOM APIs.
- DOM2Namespace - A sample application using DOM Level 2.0 APIs
- DOMRangeSample - A sample application using DOM Range APIs
- EventSample - A sample application using DOM Event APIs
- NodeIteratorSample - A sample application using DOM Iterator APIs
- TreeWalkerSample - A sample application using DOM TreeWalker APIs
- SAXSample - A sample application using SAX APIs.

- SAXNamespace - A sample application using Namespace extensions to SAX APIs.
- SAX2Namespace - A sample application using SAX 2.0
- Tokenizer - A sample application using XMLToken interface APIs.

  The Tokenizer application implements XMLToken interface, which must be registered using the setTokenHandler() method. A request for the XML tokens is registered using the setToken() method. During tokenizing, the parser doesn't validate the document and does not include or read internal/external utilities.
- DOMCompression - A sample application to compress a DOM tree
- DOMDeCompression - A sample to read back a DOM from a compressed stream
- SAXCompression - A sample application to compress the SAX output from a SAX Parser.
- SAXDeCompression - A sample application to regenerate the SAX events from the compressed stream.
- JAXPExamples - a few samples of using JAXP 1.1 API to run Oracle engine.

To run these sample programs:

- Use `make` to generate .class files.
- Add `xmlparserv2.jar` and the current directory to the `CLASSPATH`.
- Run the sample program for DOM/SAX APIs:

  ```
  java classname sample_xml_file
  ```

- Run the sample program for XSL APIs:

  ```
  java XSLSample sample_xsl_file sample_xml_file
  ```

- Run the sample program for Tokenizer APIs:

  ```
  java Tokenizer sample_xml_file token_string
  ```

- Run the sample program for compressing a DOM tree

  ```
  java DOMCompression sample.dat
  ```

  The compressed output is generated in a file called "xml.ser"
- Run the sample program to build the DOM tree from the compressed stream.

```
java DeCompression xml.ser
```

- Run the sample program for compressing the SAX events

  ```
  java SAXCompression sample.dat
  ```

- Run the sample program for regenerating the SAX events from the compressed stream.

  ```
  java SAXDeCompression xml.ser
  ```

- Run the sample program for JAXP 1.1 API

  ```
  java JAXPExamples
  ```

  the .xml and .xsl are given inside JAXPExamples.java

A few `.xml` and files are provided as test cases in directory `common`.

The XSL stylesheet `iden.xsl` can be used to achieve an identity transformation of the XML files in a `common` directory.

XML Parser for Java - XML Example 1: class.xml

```
<?xml version = "1.0"?>
<!DOCTYPE course [
<!ELEMENT course (Name, Dept, Instructor, Student)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Dept (#PCDATA)>
<!ELEMENT Instructor (Name)>
<!ELEMENT Student (Name*)>
]>
<course>
<Name>Calculus</Name>
<Dept>Math</Dept>
<Instructor>
<Name>Jim Green</Name>
</Instructor>
<Student>
<Name>Jack</Name>
<Name>Mary</Name>
<Name>Paul</Name>
</Student>
</course>
```

XML Parser for Java - XML Example 2: Using DTD employee — employee.xml

```
<?xml version="1.0"?>
<!DOCTYPE employee [
<!ELEMENT employee (Name, Dept, Title)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Dept (#PCDATA)>
<!ELEMENT Title (#PCDATA)>
]>
<employee>
<Name>John Goodman</Name>
<Dept>Manufacturing</Dept>
<Title>Supervisor</Title>
</employee>
```

XML Parser for Java - XML Example 3: Using DTD family.dtd — family.xml

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE family SYSTEM "family.dtd">
<family lastname="Smith">
<member memberid="m1">Sarah</member>
<member memberid="m2">Bob</member>
<member memberid="m3" mom="m1" dad="m2">Joanne</member>
<member memberid="m4" mom="m1" dad="m2">Jim</member>
</family>
```

DTD: family.dtd

```
<!ELEMENT family (member*)>
<!ATTLIST family lastname CDATA #REQUIRED>
<!ELEMENT member (#PCDATA)>
<!ATTLIST member memberid ID #REQUIRED>
<!ATTLIST member dad IDREF #IMPLIED>
<!ATTLIST member mom IDREF #IMPLIED>
```

XML Parser for Java - XSL Example 1: XSL (iden.xsl)

```
<?xml version="1.0"?>
<!-- Identity transformation -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="*|@*|comment()|processing-instruction()|text()">
      <xsl:copy>
          <xsl:apply-templates
select="*|@*|comment()|processing-instruction()|text()"/>
      </xsl:copy>
```

```
  </xsl:template>

</xsl:stylesheet>
```

XML Parser for Java - DTD Example 1: (NSExample)

```
<!DOCTYPE doc [
<!ELEMENT doc (child*)>
<!ATTLIST doc xmlns:nsprefix CDATA #IMPLIED>
<!ATTLIST doc xmlns CDATA #IMPLIED>
<!ATTLIST doc nsprefix: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.
</child>
</doc>
```

Using XML Parser for Java: DOMParser() Class

To write DOM based parser applications you can use the following classes:

- `DOMNamespace()` class
- `DOMParser()` class
- `XMLParser()` class

Since DOMParser extends XMLParser, all methods of XMLParser are also available to DOMParser. Figure 4–4 shows the main steps you need when coding with the `DOMParser()` class:

- *Without DTD Input*

  1. A new `DOMParser()` class is called. Available properties to use with this class are:

     * `setValidateMode`
     * `setPreserveWhiteSpace`
     * `setDocType`
     * `setBaseURL`
     * `showWarnings`

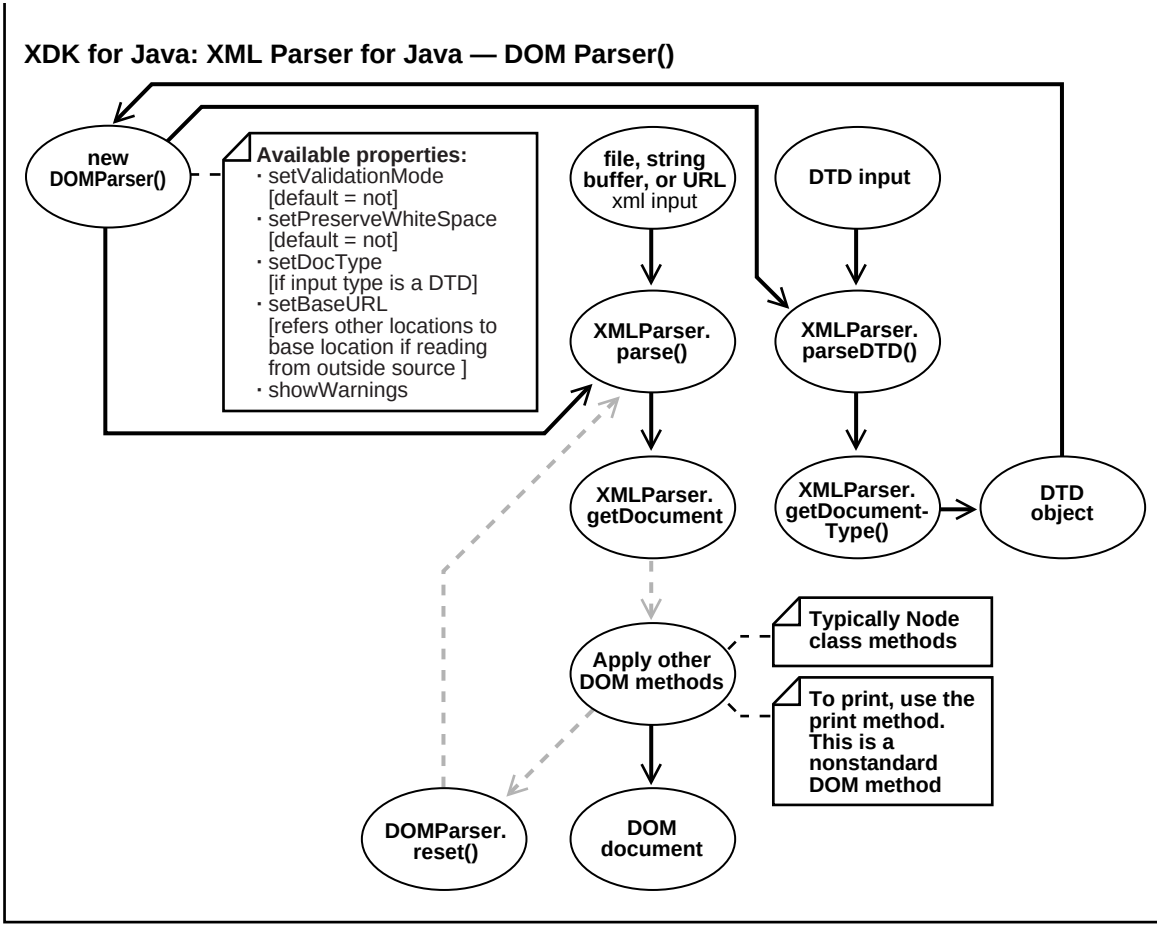
2. The results of 1) are passed to XMLParser.parse() along with the XML input. The XML input can be a file, a string buffer, or URL.
3. Use the XMLParser.getDocument() method.
4. Optionally, you can apply other DOM methods such as:
   * `print()`
   * `DOMNamespace()` methods
5. The Parser outputs the DOM tree XML (parsed) document.
6. Optionally, use `DOMParser.reset()` to clean up any internal data structures, once the Parser has finished building the DOM tree.

- *With a DTD Input*

1. A new `DOMParser()` class is called. The available properties to apply to this class are:
   * `setValidateMode`
   * `setPreserveWhiteSpace`
   * `setDocType`
   * `setBaseURL`
   * `showWarnings`
2. The results of 1) are passed to `XMLParser.parseDTD()` method along with the DTD input.
3. `XMLParser.getDocumentType()` method then sends the resulting DTD object back to the new `DOMParser()` and the process continues until the DTD has been applied.

The example, "XML Parser for Java Example 1: Using the Parser and DOM API", shows hoe to use `DOMParser()` class.

Figure 4–4 XML Parser for Java: DOMParser()



XML Parser for Java Example 1: Using the Parser and DOM API

The examples represent the way we write code so it is required to present the examples with Java coding standards (like all imports expanded), with documentation headers before the methods, and so on.

```
// This file demonstates a simple use of the parser and DOM API.
// The XML file given to the application is parsed.
// The elements and attributes in the document are printed.
// This demonstrates setting the parser options.
```

```
//

import java.io.*;
import java.net.*;
import org.w3c.dom.*;
import org.w3c.dom.Node;

import oracle.xml.parser.v2.*;

public class DOMSample
{
   static public void main(String[] argv)
   {
      try
      {
         if (argv.length != 1)
         {
            // Must pass in the name of the XML file.
            System.err.println("Usage: java DOMSample filename");
            System.exit(1);
         }

         // Get an instance of the parser
         DOMParser parser = new DOMParser();

// Generate a URL from the filename.
URL url = createURL(argv[0]);

         // Set various parser options: validation on,
         // warnings shown, error stream set to stderr.
         parser.setErrorStream(System.err);
         parser.setValidationMode(DTD_validation);
         parser.showWarnings(true);

// Parse the document.
         parser.parse(url);

         // Obtain the document.
         XMLDocument doc = parser.getDocument();

         // Print document elements
         System.out.print("The elements are: ");
         printElements(doc);

         // Print document element attributes
```

```
      System.out.println("The attributes of each element are: ");
      printElementAttributes(doc);
      parser.reset();
   }
   catch (Exception e)
   {
      System.out.println(e.toString());
   }
}

static void printElements(Document doc)
{
   NodeList nl = doc.getElementsByTagName("*");
   Node n;

   for (int i=0; i<nl.getLength(); i++)
   {
      n = nl.item(i);
      System.out.print(n.getNodeName() + " ");
   }

   System.out.println();
}

static void printElementAttributes(Document doc)
{
   NodeList nl = doc.getElementsByTagName("*");
   Element e;
   Node n;
   NamedNodeMap nnm;

   String attrname;
   String attrval;
   int i, len;

   len = nl.getLength();
   for (int j=0; j < len; j++)
   {
      e = (Element)nl.item(j);
      System.out.println(e.getTagName() + ":");
      nnm = e.getAttributes();
      if (nnm != null)
      {
         for (i=0; i<nnm.getLength(); i++)
         {
```

```
                n = nnm.item(i);
                attrname = n.getNodeName();
                attrval = n.getNodeValue();
                System.out.print(" " + attrname + " = " + attrval);
             }
          }
          System.out.println();
       }
    }

    static URL createURL(String fileName)
    {
       URL url = null;
       try
       {
          url = new URL(fileName);
       }
       catch (MalformedURLException ex)
       {
          File f = new File(fileName);
          try
          {
             String path = f.getAbsolutePath();
             String fs = System.getProperty("file.separator");
             if (fs.length() == 1)
             {
                char sep = fs.charAt(0);
                if (sep != '/')
                   path = path.replace(sep, '/');
                if (path.charAt(0) != '/')
                   path = '/' + path;
             }
             path = "file://" + path;
             url = new URL(path);
          }
          catch (MalformedURLException e)
          {
             System.out.println("Cannot create url for: " + fileName);
             System.exit(0);
          }
       }
       return url;
    }
}
```

Comments on DOMParser() Example 1

See also Figure 4–4. The following provides comments for Example 1:

1. Declare a new `DOMParser()`. In Example 1, see the line:

   ```
   DOMParser parser = new DOMParser();
   ```

   This class has several properties you can use. Here the example uses:

   ```
   parser.setErrorStream(System.err);
   parser.setValidationMode(DTD_validation);
   parser.showWarnings(true);
   ```

2. The XML input is a URL as declared by:

   ```
   URL url = createURL(argv[0])
   ```

3. The XML document is input as a URL. This is parsed using parser.parse():

   ```
   parser.parse(url);
   ```

4. Gets the document:

   ```
   XMLDocument doc = parser.getDocument();
   ```

5. Applies other DOM methods. In this case:

   - Node class methods:
     - getElementsByTagName()
     - getAttributes()
     - getNodeName()
     - getNodeValue()
   - Method, createURL() to convert the string name into a URL.

6. parser.reset() is called to clean up any data structure created during the parse process, after the DOM tree has been created. Note that this is a new method with this release.

7. Generates the DOM tree (parsed XML) document for further processing by your application.

Note: No DTD input is shown in Example 1.

Using XML Parser for Java: DOMNamespace() Class

Figure 4–3 illustrates the main processes involved when parsing an XML document using the DOM interface. The DOMNamespace() method is applied in the parser process at the "bubble" that states "Apply other DOM methods". The following example illustrates how to use DOMNamespace():

- "XML Parser for Java Example 2: Parsing a URL — DOMNamespace.java"

XML Parser for Java Example 2: Parsing a URL — DOMNamespace.java

```
// This file demonstates a simple use of the parser and Namespace
// extensions to the DOM APIs.
// The XML file given to the application is parsed and the
// elements and attributes in the document are printed.
//

import java.io.*;
import java.net.*;

import oracle.xml.parser.v2.DOMParser;

import org.w3c.dom.*;
import org.w3c.dom.Node;

// Extensions to DOM Interfaces for Namespace support.
import oracle.xml.parser.v2.XMLElement;
import oracle.xml.parser.v2.XMLAttr;


public class DOMNamespace
{
   static public void main(String[] argv)
   {
      try
      {
         if (argv.length != 1)
         {
            // Must pass in the name of the XML file.
            System.err.println("Usage: DOMNamespace filename");
            System.exit(1);
         }

         // Get an instance of the parser
         Class cls = Class.forName("oracle.xml.parser.v2.DOMParser");
```

```
            DOMParser parser = (DOMParser)cls.newInstance();

        // Generate a URL from the filename.
        URL url = createURL(argv[0]);

        // Parse the document.
            parser.parse(url);

            // Obtain the document.
            Document doc = parser.getDocument();

            // Print document elements
            printElements(doc);

            // Print document element attributes
            System.out.println("The attributes of each element are: ");
            printElementAttributes(doc);
         }
         catch (Exception e)
         {
            System.out.println(e.toString());
         }
      }

      static void printElements(Document doc)
      {
         NodeList nl = doc.getElementsByTagName("*");
         XMLElement nsElement;

         String qName;
         String localName;
         String nsName;
         String expName;

         System.out.println("The elements are: ");
         for (int i=0; i < nl.getLength(); i++)
         {
            nsElement = (XMLElement)nl.item(i);

            // Use the methods getQualifiedName(), getLocalName(), getNamespace()
            // and getExpandedName() in NSName interface to get Namespace
            // information.

            qName = nsElement.getQualifiedName();
            System.out.println("  ELEMENT Qualified Name:" + qName);
```

```
      localName = nsElement.getLocalName();
      System.out.println("  ELEMENT Local Name    :" + localName);

      nsName = nsElement.getNamespace();
      System.out.println("  ELEMENT Namespace     :" + nsName);

      expName = nsElement.getExpandedName();
      System.out.println("  ELEMENT Expanded Name :" + expName);
   }

   System.out.println();
}

static void printElementAttributes(Document doc)
{
   NodeList nl = doc.getElementsByTagName("*");
   Element e;
   XMLAttr nsAttr;
   String attrname;
   String attrval;
   String attrqname;

   NamedNodeMap nnm;
   int i, len;
   len = nl.getLength();
   for (int j=0; j < len; j++)
   {
      e = (Element) nl.item(j);
      System.out.println(e.getTagName() + ":");
      nnm = e.getAttributes();

      if (nnm != null)
      {
         for (i=0; i < nnm.getLength(); i++)
         {
            nsAttr = (XMLAttr) nnm.item(i);

            // Use the methods getQualifiedName(), getLocalName(),
            // getNamespace() and getExpandedName() in NSName
            // interface to get Namespace information.

            attrname = nsAttr.getExpandedName();
            attrqname = nsAttr.getQualifiedName();
            attrval = nsAttr.getNodeValue();
```

```
                System.out.println(" " + attrqname + "(" + attrname + ")" + " = "
+ attrval);
             }
          }
          System.out.println();
       }
    }

    static URL createURL(String fileName)
    {
       URL url = null;
       try
       {
          url = new URL(fileName);
       }
       catch (MalformedURLException ex)
       {
          File f = new File(fileName);
          try
          {
             String path = f.getAbsolutePath();
             String fs = System.getProperty("file.separator");
             if (fs.length() == 1)
             {
                char sep = fs.charAt(0);
                if (sep != '/')
                   path = path.replace(sep, '/');
                if (path.charAt(0) != '/')
                   path = '/' + path;
             }
             path = "file://" + path;
             url = new URL(path);
          }
          catch (MalformedURLException e)
          {
             System.out.println("Cannot create url for: " + fileName);
             System.exit(0);
          }
       }
       return url;
    }
}
```

Note: No DTD is input is shown in Example 2.

Using XML Parser for Java: SAXParser() Class

Applications can register a SAX handler to receive notification of various parser events. XMLReader is the interface that an XML parser's SAX2 driver must implement. This interface enables an application to set and query features and properties in the parser, to register event handlers for document processing, and to initiate a document parse.

All SAX interfaces are assumed synchronous: the parse methods must not return until parsing is complete, and readers must wait for an event-handler callback to return before reporting the next event.

This interface replaces the (now deprecated) SAX 1.0 Parser interface. The XMLReader interface contains two important enhancements over the old Parser interface:

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

Table 4–1 lists the class SAXParser() methods.

Table 4–1 Class SAXParser() Methods

Method	Description
getContentHandler()	Returns the current content handler.
getDTDHandler()	Returns the current DTD handler.
getEntityResolver()	Returns the current entity resolver.
getErrorHandler()	Returns the current error handler.
getFeature(java.lang.String name)	Looks up the value of a feature.
getProperty(java.lang.String name)	Looks up the value of a property.
setContentHandler(ContentHandler handler)	enables an application to register a content event handler.
setDocumentHandler(DocumentHandler handler)	Deprecated. as of SAX2.0 - Replaced by setContentHandler

Table 4–1 Class SAXParser() Methods(Cont.)

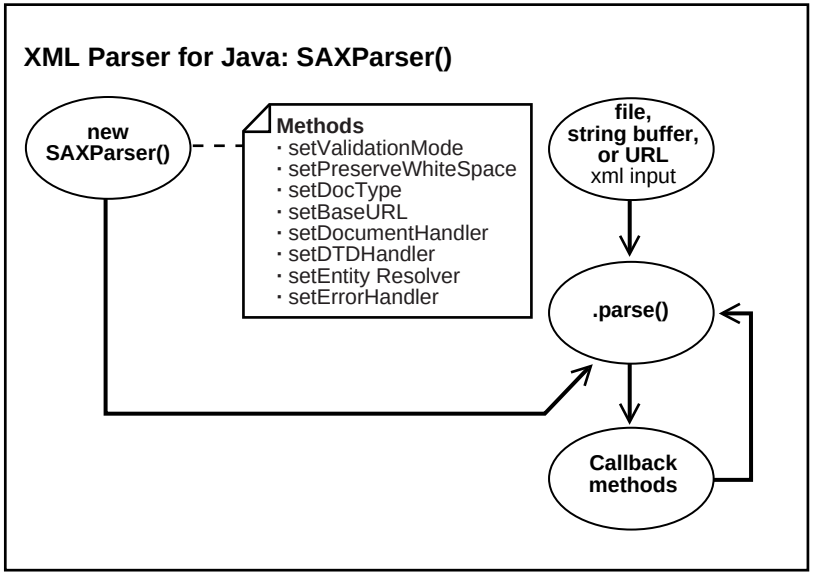
Method	Description
setDTDHandler(DTDHandler handler)	enables an application to register a DTD event handler.
setEntityResolver(EntityResolver resolver)	enables an application to register an entity resolver.
setErrorHandler(ErrorHandler handler)	enables an application to register an error event handler.
setFeature(java.lang.String name, boolean value)	Sets the state of a feature.
setProperty(java.lang.String name, java.lang.Object value)	Sets the value of a property.

Figure 4–5 shows the main steps for coding with the SAXParser() class.

1. Declare a new `SAXParser()` class. Table 4–1 lists the available methods.
2. The results of 1) are passed to .parse() along with the XML input in the form of a file, string, or URL.
3. Parse methods return when parsing completes. Meanwhile the process waits for an event-handler callback to return before reporting the next event.
4. The parsed XML document is available for further processing by your application.

The example, "XML Parser for Java Example 3: Using the Parser and SAX API (SAXSample.java)", illustrates how you can use SAXParser() class and several handler interfaces.

Figure 4–5 Using SAXParser() Class



XML Parser for Java Example 3: Using the Parser and SAX API (SAXSample.java)

```
// This file demonstates a simple use of the parser and SAX API.
// The XML file given to the application is parsed and
// prints out some information about the contents of this file.
//

import org.xml.sax.*;
import java.io.*;
import java.net.*;
import oracle.xml.parser.v2.*;

public class SAXSample extends HandlerBase
{
   // Store the locator
   Locator locator;

   static public void main(String[] argv)
   {
      try
      {
         if (argv.length != 1)
```

```
        {
           // Must pass in the name of the XML file.
           System.err.println("Usage: SAXSample filename");
           System.exit(1);
        }
        // (1) Create a new handler for the parser
        SAXSample sample = new SAXSample();

        // (2) Get an instance of the parser
        Parser parser = new SAXParser();

        // (3) Set Handlers in the parser
        parser.setDocumentHandler(sample);
        parser.setEntityResolver(sample);
        parser.setDTDHandler(sample);
        parser.setErrorHandler(sample);

        // (4) Convert file to URL and parse
        try
        {
           parser.parse(fileToURL(new File(argv[0])).toString());
        }
        catch (SAXParseException e)
        {
           System.out.println(e.getMessage());
        }
        catch (SAXException e)
        {
           System.out.println(e.getMessage());
        }
     }
     catch (Exception e)
     {
        System.out.println(e.toString());
     }
  }

  static URL fileToURL(File file)
  {
     String path = file.getAbsolutePath();
     String fSep = System.getProperty("file.separator");
     if (fSep != null && fSep.length() == 1)
        path = path.replace(fSep.charAt(0), '/');
     if (path.length() > 0 && path.charAt(0) != '/')
        path = '/' + path;
```

```
   try
   {
      return new URL("file", null, path);
   }
   catch (java.net.MalformedURLException e)
   {
      throw new Error("unexpected MalformedURLException");
   }
}

////////////////////////////////////////////////////////////////////
// (5) Sample implementation of DocumentHandler interface.
////////////////////////////////////////////////////////////////////

public void setDocumentLocator (Locator locator)
{
   System.out.println("SetDocumentLocator:");
   this.locator = locator;
}

public void startDocument()
{
   System.out.println("StartDocument");
}
public void endDocument() throws SAXException
{
   System.out.println("EndDocument");
}

public void startElement(String name, AttributeList atts)
                                                 throws SAXException
{
   System.out.println("StartElement:"+name);
   for (int i=0;i<atts.getLength();i++)
   {
      String aname = atts.getName(i);
      String type = atts.getType(i);
      String value = atts.getValue(i);

      System.out.println("   "+aname+"("+type+")"+"="+value);
   }

}
public void endElement(String name) throws SAXException
{
```

```
   System.out.println("EndElement:"+name);
}

public void characters(char[] cbuf, int start, int len)
{
   System.out.print("Characters:");
   System.out.println(new String(cbuf,start,len));
}

public void ignorableWhitespace(char[] cbuf, int start, int len)
{
   System.out.println("IgnorableWhiteSpace");
}


public void processingInstruction(String target, String data)
          throws SAXException
{
   System.out.println("ProcessingInstruction:"+target+" "+data);
}

//////////////////////////////////////////////////////////////////////
// (6) Sample implementation of the EntityResolver interface.
//////////////////////////////////////////////////////////////////////

public InputSource resolveEntity (String publicId, String systemId)
                  throws SAXException
{
   System.out.println("ResolveEntity:"+publicId+" "+systemId);
   System.out.println("Locator:"+locator.getPublicId()+" "+
               locator.getSystemId()+
               " "+locator.getLineNumber()+" "+locator.getColumnNumber());
   return null;
}

//////////////////////////////////////////////////////////////////////
// (7) Sample implementation of the DTDHandler interface.
//////////////////////////////////////////////////////////////////////

public void notationDecl (String name, String publicId, String systemId)
{
   System.out.println("NotationDecl:"+name+" "+publicId+" "+systemId);
}

public void unparsedEntityDecl (String name, String publicId,
```

```
            String systemId, String notationName)
    {
       System.out.println("UnparsedEntityDecl:"+name + " "+publicId+" "+
          systemId+" "+notationName);
    }

    ////////////////////////////////////////////////////////////////////
    // (8) Sample implementation of the ErrorHandler interface.
    ////////////////////////////////////////////////////////////////////

    public void warning (SAXParseException e)
          throws SAXException
    {
       System.out.println("Warning:"+e.getMessage());
    }

    public void error (SAXParseException e)
          throws SAXException
    {
       throw new SAXException(e.getMessage());
    }


    public void fatalError (SAXParseException e)
          throws SAXException
    {
       System.out.println("Fatal error");
       throw new SAXException(e.getMessage());
    }
}
```

XML Parser for Java Example 4: (SAXNamespace.java)

```
// This file demonstrates a simple use of the Namespace extensions to
// the SAX APIs.

import org.xml.sax.*;
import java.io.*;
import java.net.URL;
import java.net.MalformedURLException;

// Extensions to the SAX Interfaces for Namespace support.
import oracle.xml.parser.v2.XMLDocumentHandler;
import oracle.xml.parser.v2.DefaultXMLDocumentHandler;
import oracle.xml.parser.v2.NSName;
```

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

import oracle.xml.parser.v2.SAXParser;

public class SAXNamespace {
  static public void main(String[] args) {
     String fileName;

     //Get the file name
     if (args.length == 0)
     {
        System.err.println("No file Specified!!!");
        System.err.println("USAGE: java SAXNamespace <filename>");
        return;
     }
     else
     {
        fileName = args[0];
     }

     try {
        // Create handlers for the parser
        // Use the XMLDocumentHandler interface for namespace support
        // instead of org.xml.sax.DocumentHandler
        XMLDocumentHandler xmlDocHandler = new XMLDocumentHandlerImpl();

        // For all the other interface use the default provided by
        // Handler base
        HandlerBase defHandler = new HandlerBase();

        // Get an instance of the parser
        SAXParser parser = new SAXParser();

        // Set Handlers in the parser
        // Set the DocumentHandler to XMLDocumentHandler
        parser.setDocumentHandler(xmlDocHandler);

        // Set the other Handler to the defHandler
        parser.setErrorHandler(defHandler);
        parser.setEntityResolver(defHandler);
        parser.setDTDHandler(defHandler);

        try
        {
           parser.parse(fileToURL(new File(fileName)).toString());
```

```
            }
            catch (SAXParseException e)
            {
               System.err.println(args[0] + ": " + e.getMessage());
            }
            catch (SAXException e)
            {
               System.err.println(args[0] + ": " + e.getMessage());
            }
         }
         catch (Exception e)
         {
            System.err.println(e.toString());
         }
      }

   static public URL fileToURL(File file)
      {
       String path = file.getAbsolutePath();
       String fSep = System.getProperty("file.separator");
       if (fSep != null && fSep.length() == 1)
         path = path.replace(fSep.charAt(0), '/');
       if (path.length() > 0 && path.charAt(0) != '/')
         path = '/' + path;
       try {
         return new URL("file", null, path);
       }
       catch (java.net.MalformedURLException e) {
         /* According to the spec this could only happen if the file
   /* protocol were not recognized. */
         throw new Error("unexpected MalformedURLException");
       }
      }

      private SAXNamespace() throws IOException
       {
       }

   }
      /**************************************************************************
        Implementation of XMLDocumentHandler interface. Only the new
        startElement and endElement interfaces are implemented here. All other
        interfaces are implemented in the class HandlerBase.
        *************************************************************************/
```

```
class XMLDocumentHandlerImpl extends DefaultXMLDocumentHandler
{

   public void XMLDocumentHandlerImpl()
   {
   }


   public void startElement(NSName name, SAXAttrList atts) throws SAXException
   {

      // Use the methods getQualifiedName(), getLocalName(), getNamespace()
      // and getExpandedName() in NSName interface to get Namespace
      // information.
      String qName;
      String localName;
      String nsName;
      String expName;
      qName = name.getQualifiedName();
      System.out.println("ELEMENT Qualified Name:" + qName);
      localName = name.getLocalName();
      System.out.println("ELEMENT Local Name    :" + localName);

      nsName = name.getNamespace();
      System.out.println("ELEMENT Namespace     :" + nsName);

      expName = name.getExpandedName();
      System.out.println("ELEMENT Expanded Name :" + expName);

      for (int i=0; i<atts.getLength(); i++)
      {

      // Use the methods getQualifiedName(), getLocalName(), getNamespace()
      // and getExpandedName() in SAXAttrList interface to get Namespace
      // information.
         qName = atts.getQualifiedName(i);
         localName = atts.getLocalName(i);
         nsName = atts.getNamespace(i);
         expName = atts.getExpandedName(i);

         System.out.println(" ATTRIBUTE Qualified Name   :" + qName);
         System.out.println(" ATTRIBUTE Local Name       :" + localName);
         System.out.println(" ATTRIBUTE Namespace        :" + nsName);
         System.out.println(" ATTRIBUTE Expanded Name    :" + expName);
```

```
            // You can get the type and value of the attributes either
            // by index or by the Qualified Name.
            String type = atts.getType(qName);
            String value = atts.getValue(qName);

            System.out.println(" ATTRIBUTE Type            :" + type);
            System.out.println(" ATTRIBUTE Value           :" + value);
            System.out.println();
         }
      }

     public void endElement(NSName name) throws SAXException
      {
         // Use the methods getQualifiedName(), getLocalName(), getNamespace()
         // and getExpandedName() in NSName interface to get Namespace
         // information.
         String expName = name.getExpandedName();
         System.out.println("ELEMENT Expanded Name  :" + expName);
      }
   }
```

oraxml - Oracle XML parser

`oraxml` is a command-line interface to parse an XML document. It checks for well-formedness and validity.

To use oraxml ensure the following:

- Your CLASSPATH environment variable is set to point to the `xmlparserv2.jar` file that comes with Oracle XML V2 parser for Java. Because `oraxml` supports schema validation, include `xschema.jar` also in your CLASSPATH
- Your PATH environment variable can find the java interpreter that comes with JDK 1.1.x or JDK 1.2.

Use the following syntax to invoke `oraxml`:

`oraxml` *`options source`*

`oraxml` expects to be given an XML file to parse. Table 4–2 lists oraxml's command line options.

Table 4–2 oraxml: Command Line Options

Option	Purpose
-comp *fileName*	Compresses the input XML file.
-decomp *fileName*	Decompresses the input compressed file.
-dtd *fileNam*e	Validates the input file with DTD Validation.
-enc *fileName*	Prints the encoding of the input file
-help	Prints the help message.
-log *logfile*	Writes the errors / logs to the output file.
-novalidate *fileName*	Checks whether the input file is well-formed.
-schema *fileNam*e	Validates the input file with Schema Validation.
-version	Prints the release version
-warning	Show warnings.

Using JAXP

The Java API for XML Processing (JAXP) gives you the ability to use the SAX, DOM, and XSLT processors from your Java application. JAXP enables applications to parse and transform XML documents using an API that is independent of a particular XML processor implementation.

JAXP has a *pluggability* layer that enables you to plug in an implementation of a processor. The JAXP APIs have an API structure consisting of abstract classes providing a thin layer for parser pluggability. Oracle has implemented JAXP based on the Sun Microsystems reference implementation.

> **See Also:** More examples can be found at the URL
>
> `http://technet.oracle.com/tech/xml`
>
> and in the directory `xdk/demo/java/parser/jaxp`

JAXP Example: (JAVAExamples.java)

```
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.sax.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
```

```
import java.io.*;
import java.util.*;
import java.net.URL;
import java.net.MalformedURLException;

import org.xml.sax.*;
import org.xml.sax.ext.*;
import org.xml.sax.helpers.*;
import org.w3c.dom.*;

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();
         System.out.println("--- identity ---");
         identity(xmlID);
         //
         URL generalURL = createURL("general.xml");
         String generalID = generalURL.toString();
         URL ageURL = createURL("age.xsl");
         String ageID = ageURL.toString();
         System.out.println();
         System.out.println("--- namespaceURI ---");
         namespaceURI(generalID, ageID);
         //
         System.out.println();
         System.out.println("--- templatesHandler ---");
         templatesHandler(xmlID, xslID);
         System.out.println();
         System.out.println("--- contentHandler2contentHandler ---");
         contentHandler2contentHandler(xmlID, xslID);
         System.out.println();
```

```
            System.out.println("--- contentHandler2DOM ---");
            contentHandler2DOM(xmlID, xslID);
            System.out.println();
            System.out.println("--- reader ---");
            reader(xmlID, xslID);
            System.out.println();
            System.out.println("--- xmlFilter ---");
            xmlFilter(xmlID, xslID);
            //
            URL xslURLtwo = createURL("jaxptwo.xsl");
            String xslIDtwo = xslURLtwo.toString();
            URL xslURLthree = createURL("jaxpthree.xsl");
            String xslIDthree = xslURLthree.toString();
            System.out.println();
            System.out.println("--- xmlFilterChain ---");
            xmlFilterChain(xmlID, xslID, xslIDtwo, xslIDthree);
         } 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));
      }
      //
      public static void identity(String xmlID)
           throws TransformerException, TransformerConfigurationException
           {
           TransformerFactory tfactory = TransformerFactory.newInstance();
         Transformer transformer = tfactory.newTransformer();
         transformer.setOutputProperty(OutputKeys.METHOD, "html");
         transformer.setOutputProperty(OutputKeys.INDENT, "no");
         StreamSource source = new StreamSource(xmlID);
              transformer.transform(source, new StreamResult(System.out));
           }
      //
      public static void namespaceURI(String xmlID, String xslID)
         throws TransformerException, TransformerConfigurationException
      {
```

```
      TransformerFactory tfactory = TransformerFactory.newInstance();
      Transformer transformer
         = tfactory.newTransformer(new StreamSource(xslID));
      System.out.println("default: 99");
      transformer.transform( new StreamSource(xmlID),
         new StreamResult(System.out));
      transformer.setParameter("{http://www.oracle.com/ages}age", "20");
      System.out.println();
      System.out.println("should say: 20");
      transformer.transform( new StreamSource(xmlID),
         new StreamResult(System.out));
      transformer.setParameter("{http://www.oracle.com/ages}age", "40");
      transformer.setOutputProperty(OutputKeys.METHOD, "html");
      System.out.println();
      System.out.println("should say: 40");
      transformer.transform( new StreamSource(xmlID),
         new StreamResult(System.out));
   }
   //
   public static void templatesHandler(String xmlID, String xslID)
      throws TransformerException, TransformerConfigurationException,
      IOException, SAXException,
      ParserConfigurationException, FileNotFoundException
   {
      TransformerFactory tfactory = TransformerFactory.newInstance();
      if (tfactory.getFeature(SAXTransformerFactory.FEATURE))
      {
         SAXTransformerFactory stfactory = (SAXTransformerFactory) tfactory;
         TemplatesHandler handler = stfactory.newTemplatesHandler();
         handler.setSystemId(xslID);
         // JDK 1.1.8
         Properties driver = System.getProperties();
         driver.put("org.xml.sax.driver", "oracle.xml.parser.v2.SAXParser");
         System.setProperties(driver);
         /** JDK 1.2.2
         System.setProperty("org.xml.sax.driver",
"oracle.xml.parser.v2.SAXParser");
         */
         XMLReader reader = XMLReaderFactory.createXMLReader();
         reader.setContentHandler(handler);
         reader.parse(xslID);
         Templates templates = handler.getTemplates();
         Transformer transformer = templates.newTransformer();
         transformer.transform(new StreamSource(xmlID), new
StreamResult(System.out));
```

```
      }
   }
   //
   public static void reader(String xmlID, String xslID)
      throws TransformerException, TransformerConfigurationException,
      SAXException, IOException, ParserConfigurationException
   {
      TransformerFactory tfactory = TransformerFactory.newInstance();
      SAXTransformerFactory stfactory = (SAXTransformerFactory)tfactory;
      StreamSource streamSource = new StreamSource(xslID);
      XMLReader reader = stfactory.newXMLFilter(streamSource);
      ContentHandler contentHandler = new oraContentHandler();
      reader.setContentHandler(contentHandler);
      InputSource is = new InputSource(xmlID);
      reader.parse(is);
   }
   //
   public static void xmlFilter(String xmlID, String xslID)
      throws TransformerException, TransformerConfigurationException,
      SAXException, IOException, ParserConfigurationException
   {
      TransformerFactory tfactory = TransformerFactory.newInstance();
      XMLReader reader = null;
      try {
              javax.xml.parsers.SAXParserFactory factory=
                 javax.xml.parsers.SAXParserFactory.newInstance();
              factory.setNamespaceAware(true);
         javax.xml.parsers.SAXParser jaxpParser=
                 factory.newSAXParser();
              reader = jaxpParser.getXMLReader();
      } catch(javax.xml.parsers.ParserConfigurationException ex) {
        throw new org.xml.sax.SAXException(ex);
      } catch(javax.xml.parsers.FactoryConfigurationError ex1) {
        throw new org.xml.sax.SAXException(ex1.toString());
      } catch(NoSuchMethodError ex2) {
      }
      if (reader == null)
         reader = XMLReaderFactory.createXMLReader();
      XMLFilter filter
         = ((SAXTransformerFactory) tfactory).newXMLFilter(new
StreamSource(xslID));
      filter.setParent(reader);
      filter.setContentHandler(new oraContentHandler());
      filter.parse(new InputSource(xmlID));
   }
```

```
//
public static void xmlFilterChain(
   String xmlID, String xslID_1,
   String xslID_2, String xslID_3)
   throws TransformerException, TransformerConfigurationException,
   SAXException, IOException
{
   TransformerFactory tfactory = TransformerFactory.newInstance();
   if (tfactory.getFeature(SAXSource.FEATURE))
   {
      SAXTransformerFactory stf = (SAXTransformerFactory)tfactory;
      XMLReader reader = null;
      try {
              javax.xml.parsers.SAXParserFactory factory =
           javax.xml.parsers.SAXParserFactory.newInstance();
         factory.setNamespaceAware(true);
              javax.xml.parsers.SAXParser jaxpParser =
                 factory.newSAXParser();
              reader = jaxpParser.getXMLReader();
      } catch(javax.xml.parsers.ParserConfigurationException ex) {
              throw new org.xml.sax.SAXException( ex );
      } catch(javax.xml.parsers.FactoryConfigurationError ex1) {
              throw new org.xml.sax.SAXException( ex1.toString() );
      } catch(NoSuchMethodError ex2) {
      }
      if (reader == null ) reader = XMLReaderFactory.createXMLReader();
      XMLFilter filter1 = stf.newXMLFilter(new StreamSource(xslID_1));
      XMLFilter filter2 = stf.newXMLFilter(new StreamSource(xslID_2));
      XMLFilter filter3 = stf.newXMLFilter(new StreamSource(xslID_3));
      if (filter1 != null && filter2 != null && filter3 != null)
      {
         filter1.setParent(reader);
         filter2.setParent(filter1);
         filter3.setParent(filter2);
         filter3.setContentHandler(new oraContentHandler());
         filter3.parse(new InputSource(xmlID));
      }
   }
}
//
public static void contentHandler2contentHandler(String xmlID, String xslID)
   throws TransformerException,
   TransformerConfigurationException,
   SAXException, IOException
{
```

```
      TransformerFactory tfactory = TransformerFactory.newInstance();

      if (tfactory.getFeature(SAXSource.FEATURE))
      {
         SAXTransformerFactory stfactory = ((SAXTransformerFactory) tfactory);
         TransformerHandler handler
            = stfactory.newTransformerHandler(new StreamSource(xslID));
         Result result = new SAXResult(new oraContentHandler());
         handler.setResult(result);
         XMLReader reader = null;
         try {
                javax.xml.parsers.SAXParserFactory factory=
                   javax.xml.parsers.SAXParserFactory.newInstance();
            factory.setNamespaceAware(true);
                javax.xml.parsers.SAXParser jaxpParser=
                   factory.newSAXParser();
            reader=jaxpParser.getXMLReader();
         } catch( javax.xml.parsers.ParserConfigurationException ex ) {
                throw new org.xml.sax.SAXException( ex );
         } catch( javax.xml.parsers.FactoryConfigurationError ex1 ) {
                throw new org.xml.sax.SAXException( ex1.toString() );
         } catch( NoSuchMethodError ex2 ) {
         }
         if( reader == null ) reader = XMLReaderFactory.createXMLReader();
         reader.setContentHandler(handler);
         reader.setProperty("http://xml.org/sax/properties/lexical-handler",
handler);
         InputSource inputSource = new InputSource(xmlID);
         reader.parse(inputSource);
      }
   }
   //
   public static void contentHandler2DOM(String xmlID, String xslID)
      throws TransformerException, TransformerConfigurationException,
      SAXException, IOException, ParserConfigurationException
   {
      TransformerFactory tfactory = TransformerFactory.newInstance();

      if (tfactory.getFeature(SAXSource.FEATURE)
         && tfactory.getFeature(DOMSource.FEATURE))
      {
         SAXTransformerFactory sfactory = (SAXTransformerFactory) tfactory;

         DocumentBuilderFactory dfactory
            = DocumentBuilderFactory.newInstance();
```

```
            DocumentBuilder docBuilder = dfactory.newDocumentBuilder();
            org.w3c.dom.Document outNode = docBuilder.newDocument();

            TransformerHandler handler
               = sfactory.newTransformerHandler(new StreamSource(xslID));
            handler.setResult(new DOMResult(outNode));

            XMLReader reader = null;

            try {
                    javax.xml.parsers.SAXParserFactory factory =
                       javax.xml.parsers.SAXParserFactory.newInstance();
                    factory.setNamespaceAware(true);
                    javax.xml.parsers.SAXParser jaxpParser=
                    factory.newSAXParser();
                    reader = jaxpParser.getXMLReader();
            } catch(javax.xml.parsers.ParserConfigurationException ex) {
                    throw new org.xml.sax.SAXException(ex);
            } catch(javax.xml.parsers.FactoryConfigurationError ex1) {
                    throw new org.xml.sax.SAXException(ex1.toString());
            } catch(NoSuchMethodError ex2) {
            }
            if(reader == null ) reader = XMLReaderFactory.createXMLReader();
            reader.setContentHandler(handler);
            reader.setProperty("http://xml.org/sax/properties/lexical-handler",
handler);
            reader.parse(xmlID);
            printDOMNode(outNode);
         }
      }
      //
      private static void printDOMNode(Node node)
         throws TransformerException, TransformerConfigurationException,
SAXException, IOException,
         ParserConfigurationException
      {
         TransformerFactory tfactory = TransformerFactory.newInstance();
         Transformer serializer = tfactory.newTransformer();
         serializer.setOutputProperty(OutputKeys.METHOD, "xml");
         serializer.setOutputProperty(OutputKeys.INDENT, "no");
         serializer.transform(new DOMSource(node),
            new StreamResult(System.out));
      }
      //
      private static URL createURL(String fileName)
```

```
  {
     URL url = null;
     try
     {
        url = new URL(fileName);
     }
     catch (MalformedURLException ex)
     {
        File f = new File(fileName);
        try
        {
           String path = f.getAbsolutePath();
           // This is a bunch of weird code that is required to
           // make a valid URL on the Windows platform, due
           // to inconsistencies in what getAbsolutePath returns.
           String fs = System.getProperty("file.separator");
           if (fs.length() == 1)
           {
              char sep = fs.charAt(0);
              if (sep != '/')
                 path = path.replace(sep, '/');
              if (path.charAt(0) != '/')
                 path = '/' + path;
           }
           path = "file://" + path;
           url = new URL(path);
        }
        catch (MalformedURLException e)
        {
           System.out.println("Cannot create url for: " + fileName);
           System.exit(0);
        }
     }
     return url;
  }
}
```

JAXP Example: (oraContentHandler.java

```
import org.xml.sax.ContentHandler;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.Locator;

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");
   }

   public void startPrefixMapping(String prefix, String uri)
      throws SAXException
   {
      System.out.println(TRADE_MARK + "- startPrefixMapping: "
         + prefix + ", " + uri);
   }

   public void endPrefixMapping(String prefix)
      throws SAXException
   {
      System.out.println(TRADE_MARK + " - endPrefixMapping: "
         + prefix);
   }

   public void startElement(String namespaceURI, String localName,
      String qName, Attributes atts)
      throws SAXException
   {
      System.out.print(TRADE_MARK + "- startElement: "
         + namespaceURI + ", " + namespaceURI +
         ", " + qName);
      int n = atts.getLength();
      for(int i = 0; i < n; i++)
         System.out.print(", " + atts.getQName(i));
      System.out.println("");
```

```
    }

    public void endElement(String namespaceURI, String localName,
       String qName)
       throws SAXException
    {
       System.out.println(TRADE_MARK + "- endElement: "
          + namespaceURI + ", " + namespaceURI
          + ", " + qName);
    }

    public void characters(char ch[], int start, int length)
       throws SAXException
    {
       String s = new String(ch, start, (length > 30) ? 30 : length);
       if(length > 30)
          System.out.println(TRADE_MARK + "- characters: \""
             + s + "\"...");
       else
          System.out.println(TRADE_MARK + "- characters: \""
             + s + "\"");
    }

    public void ignorableWhitespace(char ch[], int start, int length)
       throws SAXException
    {
       System.out.println(TRADE_MARK + "- ignorableWhitespace");
    }

    public void processingInstruction(String target, String data)
       throws SAXException
    {
       System.out.println(TRADE_MARK + "- processingInstruction: "
          + target + ", " + target);
    }

    public void skippedEntity(String name)
       throws SAXException
    {
       System.out.println(TRADE_MARK + "- skippedEntity: " + name);
    }
}
```

Frequently Asked Questions About DTDs

This section lists DTD questions and answers.

Why Can't My Parser Find the DTD File?

Answer: The DTD file defined in the `<!DOCTYPE>` declaration must be relative to the location of the input XML document. Otherwise, you'll need to use the `setBaseURL(url)` functions to set the base URL to resolve the relative address of the DTD if the input is coming from InputStream.

Can I Validate an XML File Using an External DTD?

Answer: You need to include a reference to the applicable DTD in your XML document. Without it there is no way for the parser to know what to validate against. Including the reference is the XML standard way of specifying an external DTD. Otherwise you need to embed the DTD in your XML Document.

Does Oracle Perform DTD Caching?

Do you have DTD caching? How do I set the DTD using version 2 of the parser for DTD Cache purpose?

Answer: Yes, DTD caching is optional and is not enabled automatically.

The method to set the DTD is `setDoctype()`. Here is an example:

```
// Test using InputSource
parser = new DOMParser();
parser.setErrorStream(System.out);
parser.showWarnings(true);

FileReader r = new FileReader(args[0]);
InputSource inSource = new InputSource(r);
inSource.setSystemId(createURL(args[0]).toString());
parser.parseDTD(inSource, args[1]);
dtd = (DTD)parser.getDoctype();

r = new FileReader(args[2]);
inSource = new InputSource(r);
inSource.setSystemId(createURL(args[2]).toString());
// ********************
parser.setDoctype(dtd);
// ********************
parser.setValidationMode(DTD_validation);
```

```
parser.parse(inSource);

doc = (XMLDocument)parser.getDocument();
doc.print(new PrintWriter(System.out));
```

How Does the XML Parser for Java Recognize External DTDs?

How does the XML Parser for Java version 2 recognize external DTDs when running from the server? The Java code has been loaded with loadjava and runs in the Oracle9*i* server process. My XML file has an external DTD reference.

1. Is there a generic way, as there is with the SAX parser, to redirect it to a stream or string or something if my DTD is in the database?
2. Is there a generic way to redirect the DTD, as there is with the SAX parser, with `resolveEntity()`?

Answer:

1. We only have the `setBaseURL()` method at this time.
2. You can achieve your desired result using the following:
   a. Parse your External DTD using a DOM parser's `parseDTD()` method.
   b. Call `getDoctype()` to get an instance of `oracle.xml.parser.v2.DTD`.
   c. On the document where you want to set your DTD programmatically, use the `setDoctype(yourDTD)`. We use this technique to read a DTD out of our product's JAR file.

How Do I Load External DTDs from a JAR File?

I would like to put all my DTDs in a JAR file, so that when the XML parser needs a DTD it can get it from the JAR. The current XML parser supports a base URL (`setBaseURL()`), but that just points to a place where all the DTDs are exposed.

Answer: The solution involves the following steps:

1. Load the DTD as an InputStream using:

   ```
   InputStream is =
   YourClass.class.getResourceAsStream("/foo/bar/your.dtd");
   ```

   This will open `./foo/bar/your.dtd` in the first relative location on the `CLASSPATH` that it can be found, including out of your JAR if it's in the `CLASSPATH`.

2. Parse the DTD with the code:

```
DOMParser d = new DOMParser();
d.parseDTD(is, "rootelementname");
d.setDoctype(d.getDoctype());
```

3. Now parse your document with the following code:

```
d.parse("yourdoc");
```

Can I Check the Correctness of an XML Document Using Their DTD?

I am exporting Java objects to XML. I can construct a DOM with an XML document and use its print method to export it. However, I am unable to set the DTD of these documents. I construct a parser, parse the DTD, and then get the DTD through document `doc = parser.getDocument()` and DocType `dtd = doc.getDocumentType()`.

How do I set the DTD of the freshly constructed XML documents to use this one in order to be able to check the correctness of the documents at a later time?

Answer: Your method of getting the DTD object is correct. However, we do not do any validation while creating the DOM tree using DOM APIs. So setting the DTD in the document will not help validate the DOM tree that is constructed. The only way to validate an XML file is to parse the XML document using the DOM parser or the SAX parser.

How Do I Parse a DTD Object Separately from My XML Document?

How do I parse and get a DTD object separately from parsing my XML document?

Answer: The `parseDTD()` method enables you to parse a DTD file separately and get a DTD object. Here is a sample code to do that:

```
DOMParser domparser = new DOMParser();
domparser.setValidationMode(DTD_validation);
/* parse the DTD file */
domparser.parseDTD(new FileReader(dtdfile));
DTD dtd = domparser.getDocType();
```

Is the XML Parser Case-Sensitive?

The XML file has a tag like: `<xn:subjectcode>`. In the DTD, it is defined as `<xn:subjectCode>`. When the file is parsed and validated against the DTD, it

returns the error: `XML-0148: (Error) Invalid element 'xn:subjectcode' in content of 'xn:Resource',...`

When I changed the element name to `<xn:subjectCode>` instead of `<xn:subjectcode>` it works. Is the parser case-sensitive as far as validation against DTDs go - or is it because, there is a namespace also in the tag definition of the element and when a element is defined along with its namespace, the case-sensitivity comes into effect?

Answer: XML is inherently case-sensitive, therefore our parsers enforce case sensitivity in order to be compliant. When you run in non-validation mode only well-formedness counts. However `<test></Test>` would signal an error even in non-validation mode.

How Do I Extract Embedded XML from a CDATA Section?

Given:

```
<PAYLOAD>
<![CDATA[<?xml version = '1.0' encoding = 'ASCII' standalone = 'no'?>
<ADD_PO_003>
   <CNTROLAREA>
      <BSR>
         <VERB value="ADD">ADD</VERB>
         <NOUN value="PO">PO</NOUN>
         <REVISION value="003">003</REVISION>
      </BSR>
   </CNTROLAREA>
</ADD_PO_003>]]>
</PAYLOAD>
```

1. How do I extract `PAYLOAD` to do extra processing on it?
2. When I select the value of `PAYLOAD` it does not parse the data because it is in a CDATA section. Why?
3. How do I extract embedded XML using just XSLT? I have done this using SAX before but in the current setup all I can use is XSLT.

Answer:

1. Here are the answers:

   The CDATA strategy is kind of odd. You won't be able to use a different encoding on the nested XML document included as text inside the CDATA, so having the XML declaration of the embedded document seems of little value to

me. If you don't need the XML declaration, then why not just embed the message as real elements into the <PAYLOAD> instead of as a text chunk which is what CDATA does for you.

Just use the following code:

```
String s = YourDocumentObject.selectSingleNode("/OES_MESSAGE/PAYLOAD");
```

2. It shouldn't parse the data, you've asked for it to be a big text chunk, which is what it will give you. You'll have to parse the text chunk yourself (another benefit of not using the CDATA approach) by doing something like:

```
YourParser.parse( new StringReader(s));
```

where s is the string you got in the previous step.

3. There is nothing special about the content of your CDATA, it's just text. If you want the text content to be output without escaping the angle-brackets, then you'll do:

```
<xsl:value-of select="/OES_MESSAGE/PAYLOAD" disable-output-escaping="yes"/>
```

Why Am I Getting an Error When I Call DOMParser.parseDTD()?

I am having trouble creating a DTD and parsing it using Oracle XML Parser for Java version 2. I got the following error when I call `DOMParser.parseDTD()` function:

```
Attribute value should start with quote.
```

Please check my DTD and tell me what's wrong.

```
<?xml version = "1.0" encoding="UTF-8" ?>
<!-- RCS_ID = "$Header: XMLRenderer.dtd 115.0 2000/09/18 03:00:10 fli noship $"
-->
<!-- RCS_ID_RECORDED = VersionInfo.recordClassVersion(RCS_ID,
"oracle.apps.mwa.admin") -->
<!--  Copyright: This DTD file is owned by Oracle Mobile Application Server
Group.  -->
  <!ELEMENT    page    (header?,form,footer?) >
  <!ATTLIST    page
               name    CDATA   #REQUIRED
               lov     (Y|N)   'N' >
  <!ELEMENT    header EMPTY >
  <!ATTLIST    header
               name    CDATA   #REQUIRED
               title   CDATA
               home    (Y|N)   'N'
```

```
              portal  (Y|N)   'N'
              logout  (Y|N)   'N' >
  <!ELEMENT   footer EMPTY >
  <!ATTLIST   footer
              name    CDATA   #REQUIRED
              home    (Y|N)   'N'
              portal  (Y|N)   'N'
              logout  (Y|N)   'N'
              copyright (Y|N) 'N' >

  <!ELEMENT   form
  (styledText|textInput|list|link|menu|submitButton|table|separator)+ >
  <!ATTLIST   form
              name    CDATA    #REQUIRED
              title    CDATA
              type     CDATA >

  <!ELEMENT   styledText    (#PCDATA) >

  <!ELEMENT   textInput    EMPTY >
  <!ATTLIST   textInput
              name    CDATA    #REQUIRED
              prompt    CDATA    #IMPLIED
              password    (Y|N)    'N'
              required    (Y|N)    'N'
              maxlength    #IMPLIED
              size    #IMPLIED
              format    #IMPLIED
              default    #IMPLIED >

  <!ELEMENT   link (postfield*) >
  <!ATTLIST   link
              name    CDATA    #REQUIRED
              title    CDATA    #REQUIRED
              baseurl    CDATA    #REQUIRED >
```

Answer: Your DTD syntax is not valid. When you declare `ATTLIST` with CDATA, you must put `#REQUIRED`, `#IMPLIED`, `#FIXED`, `"any value"`, or`%paramatic_entity`. For example, your DTD contains:

```
<!ELEMENT  header EMPTY >
<!ATTLIST  header
           name    CDATA   #REQUIRED
           title   CDATA
           home    (Y|N)   'N'
```

```
          portal  (Y|N)   'N'
          logout  (Y|N)   'N' >
```

should change as follows:

```
<!ELEMENT  header EMPTY >
<!ATTLIST  header
          name   CDATA #REQUIRED
          title  CDATA #REQUIRED<!--can be replaced by #FIXED, #IMPLIED, or
"title1" -->
          home    (Y|N)   'N'
          portal  (Y|N)   'N'
          logout  (Y|N)   'N' >
```

Is There a Standard Extension for External Entity References in an XML Document?

Is there a standard extension (other than `.xml` or `.txt`) that should be used for external entities referenced in an XML document? These external entities are not complete XML files, but rather only part of an XML file, starting with the `<![CDATA[` designation. Mostly they contain HTML, or Javascript code, but may also contain just some plain text. As an example, the external entity is `A.txt` which is being referenced in the XML document `B.xml`.

`A.txt` looks like this:

```
<![CDATA[<!-- This is just an html comment -->]]>
```

`B.xml` looks like this:

```
 <?xml version="1.0"?>
 <!DOCTYPE B[
 <!ENTITY htmlComment SYSTEM "A.txt">
]>

<B>
  &htmlComment;
</B>
```

Currently we are using `.txt` as an extension for all such entities, but need to change that, otherwise the translation team assumes that these files need to get translated, whereas they don't. Is there a standard extension that we should be using?

Answer: The file extension for external entities is unimportant so you can change it to any convenient extension, including no extension.

Frequently Asked Questions About DOM and SAX APIs

How Do I Use the DOM API to Count Tagged Elements?

How do I get the number of elements in a particular tag using the parser?

Answer: You can use the `getElementsByTagName()` method that returns a node list of all descent elements with a given tag name. You can then find out the number of elements in that node list to determine the number of the elements in the particular tag.

How Does the DOM Parser Work?

Answer: The parser accepts an XML-formatted document and constructs in memory a DOM tree based on its structure. It will then check whether the document is well-formed and optionally whether it complies with a DTD. It also provides methods to support DOM Level 1 and 2.

How Do I Create a Node with a Value to Be Set Later?

Answer: If you check the DOM spec referring to the table discussing the node type, you will find that if you are creating an element node, its node value is null and hence cannot be set. However, you can create a text node and append it to the element node. You can then put the value in the text node.

How Do I Traverse the XML Tree?

How to traverse the XML tree

Answer: You can traverse the tree by using the DOM API. Alternately, you can use the `selectNodes()` method which takes XPath syntax to navigate through the `XML document. selectNodes()` is part of `oracle.xml.parser.v2.XMLNode`.

How Do I Extract Elements from an XML File?

How do I extract elements from the XML file?

Answer: If you're using DOM, the `getElementsByTagName()` method can be used to get all of the elements in the document.

Does a DTD Validate the DOM Tree?

If I add a DTD to an XML document, does it validate the DOM tree?

Answer: No, we do not do any validation while creating the DOM tree using the DOM APIs. So setting the DTD in the document will not help in validating the DOM tree that is constructed. The only way to validate an XML file is to parse the XML document using the DOM parser or SAX parser. Set the validation mode of the parser using `setValidationMode()`.

How Do I Find the First Child Node Element Value?

How do I efficiently obtain the value of first child node of the element without going through the DOM tree?

Answer: If you do not need the entire tree, use the SAX interface to return the desired data. Since it is event-driven, it does not have to parse the whole document.

How Do I Create DocType Node?

How do I create a DocType node?

Answer: The only current way of creating a doctype node is by using the parseDTD functions. For example, emp.dtd has the following DTD:

```
<!ELEMENT employee (Name, Dept, Title)>
 <!ELEMENT Name (#PCDATA)>
<!ELEMENT Dept (#PCDATA)>
 <!ELEMENT Title (#PCDATA)>
```

You can use the following code to create a doctype node:

```
parser.parseDTD(new FileInputStream(emp.dtd), "employee");
dtd = parser.getDocType();
```

How Do I Use the XMLNode.selectNodes() Method?

How do I use the `selectNodes()` method in `XMLNode` class?

Answer: The `selectNodes()` method is used in `XMLElement` and `XMLDocument` nodes. This method is used to extract contents from the tree or subtree based on the select patterns allowed by XSL. The optional second parameter of `selectNodes`, is used to resolve namespace prefixes (that is, it returns the expanded namespace URL given a prefix). `XMLElement` implements `NSResolver`, so it can be sent as the second parameter. `XMLElement` resolves the prefixes based on the input document.

You can use the `NSResolver` interface, if you need to override the namespace definitions. The following sample code uses `selectNodes`

```
public class SelectNodesTest  {
public static void main(String[] args) throws Exception {
String pattern = "/family/member/text()";
String file    = args[0];

if (args.length == 2)
  pattern = args[1];

DOMParser dp = new DOMParser();

dp.parse(createURL(file));  // Include createURL from DOMSample
XMLDocument xd = dp.getDocument();
XMLElement e = (XMLElement) xd.getDocumentElement();
NodeList nl = e.selectNodes(pattern, e);
for (int i = 0; i < nl.getLength(); i++) {
   System.out.println(nl.item(i).getNodeValue());
    }
  }
}

> java SelectNodesTest family.xml
Sarah
Bob
Joanne
Jim

> java SelectNodesTest family.xml //member/@memberid
m1
m2
m3
m4
```

How Does the SAX API Determine the Data Value?

I am using the SAX parser to parse an XML document. How does it get the value of the data?

Answer: During a SAX parse the value of an element will be the concatenation of the characters reported from after the `startElement` event to before the corresponding `endElement` event is called.

How Does SAXSample.java Call Methods?

Inside the SAXSample program, I did not see any line that explicitly calls `setDocumentLocator` and some other methods. However, these methods are run. Can you explain when they are called and from where?

Answer: SAX is a standard interface for event-based XML parsing. The parser reports parsing events directly through callback functions such as `setDocumentLocator()` and `startDocument()`. The application, in this case, the SAXSample, uses handlers to deal with the different events. The following Web site is a good place to help you start learning about the event-driven API, SAX: `http://www.megginson.com/SAX/index.html`

Does the DOMParser Use the org.xml.sax.Parser Interface?

Does the XML Parser DOMParser implement `org.xml.sax.Parser` interface? The documentation says it uses XML constants and the API does not include that class at all.

Answer: You'll want `oracle.xml.parser.v2.SAXParser` to work with SAX and to have something that implements the `org.xml.sax.Parser` interface.

How Do I Create a New Document Type Node with DOM API?

I am trying to create a XML file on the fly. I use the NodeFactory to construct a document using `createDocument()`. I have then `setStandalone("no")` and `setVersion("1.0")`. When I try to add a `DOCTYPE` node with `appendChild(new XMLNode("test", Node.DOCUMENT_TYPE_NODE))`, I get a `ClassCastException`. How do I add a node of this type? I noticed that the NodeFactory did not have a method for creating a `DOCTYPE` node.

Answer: There is no way to create a new `DOCUMENT_TYPE_NODE` object using the DOM APIs. The only way to get a DTD object is to parse the DTD file or the XML file using the DOM parser, and then use the `getDocType()` method.

Note that new `XMLNode("test",Node.DOCUMENT_TYPE_NODE)` does not create a DTD object. It creates an XMLNode object with the type set to `DOCUMENT_TYPE_NODE`, which in fact should not be allowed. The `ClassCastException` is raised because `appendChild` expects a DTD object (based on the type).

Also, we do not do any validation while creating the DOM tree using the DOM APIs. So setting the DTD in the document will not help in validating the DOM tree

that is constructed. The only way to validate an XML file is to parse the XML document using the DOM parser or the SAX Parser.

How Do I Query for First Child Node's Value of a Certain Tag?

I am using the XML Parser for Java version 2. I want to obtain the value of first child node value of a tag. I could not find any method that can do that efficiently. The nearest match is method getElementsByTag("Name"), which traverses the entire tree under.

Answer: Your best bet, if you do not need the entire tree, is to use the SAX interface to return the desired data. Since it is event driven it does not have to parse the whole document.

Can I Generate an XML Document from Data in Variables?

Is there an example of XML document generation starting from information contained in simple variables? For example, a client fills a Java form and wants to obtain an XML document containing the given data.

Answer: Here are two possible interpretations of your question and answers to both. Let's say you have two variables in Java:

```
String firstname = "Gianfranco";
String lastname = "Pietraforte";
```

The two ways to get this information into an XML document are as follows:

1. Make an XML document in a string and parse it.

```
String xml = "<person><first>"+firstname+"</first>"+
     "<last>"+lastname+"</last></person";
DOMParser d = new DOMParser();
d.parse( new StringReader(xml));
Document xmldoc = d.getDocument();
```

2. Use DOM APIs to construct the document and append it together:

```
Document xmldoc = new XMLDocument();
Element e1 = xmldoc.createElement("person");
xmldoc.appendChild(e1);
Element e2 = xmldoc.createElement("first");
e1.appendChild(e2);
Text t = xmldoc.createText(firstname);
e2.appendChild(t);
// and so on
```

How Do I Use the DOM API to Print Data in the Element Tags?

Can you suggest how to get a print out using the DOM API in Java:

<name>macy</name>

I want to print out "macy". Don't know which class and what function to use. I was successful in printing "name" on to the console.

Answer: For DOM, you need to first realize that <name>macy</name> is actually an element named "name" with a child node (Text Node) of value "macy".

So, you can do the following:

```
String value = myElement.getFirstChild().getNodeValue();
```

How Do I Build XML Files from Hash Table Value Pairs?

We have a hash table of key value pairs, how do we build an XML file out of it using the DOM API? We have a hashtable key = value name = george zip = 20000. How do we build this?

<key>value</key><name>george</name><zip>20000</zip>'

Answer:

1. Get the enumeration of keys from your hash table.
2. Loop while `enum.hasMoreElements()`.
3. For each key in the enumeration, use the `createElement()` on DOM document to create an element by the name of the key with a child text node with the value of the *value* of the hash table entry for that key.

XML Parser for Java: WRONG_DOCUMENT_ERR on Node.appendChild()

I have a question regarding our XML parser (version 2) implementation. I have the following scenario:

```
Document doc1 = new XMLDocument();
Element element1 = doc1.creatElement("foo");
Document doc2 = new XMLDocument();
Element element2 = doc2.createElement("bar");
element1.appendChild(element2);
```

My question is whether or not we should get a DOM exception of `WRONG_DOCUMENT_ERR` on calling the `appendChild()` routine.

Answer: Yes, you should get this error, since the owner document of `element1` is `doc1` while that of `element2` is `doc2`. `AppendChild()` only works within a single tree and you are dealing with two different ones.

Will WRONG_DOCUMENT_ERR Result from This Code Fragment?

In `XSLSample.java` that's shipped with the XML parser version 2:

```
DocumentFragment result = processor.processXSL(xsl, xml);
// create an output document to hold the result
  out = new XMLDocument();
// create a dummy document element for the output document
  Element root = out.createElement("root");
  out.appendChild(root);
// append the transformed tree to the dummy document element
   root.appendChild(result);
```

Nodes root and result are created from different XML documents. Wouldn't this result in the `WRONG_DOCUMENT_ERR` when we try to append result to root?

Answer: This sample uses a document fragment that does not have a root node, therefore there are not two XML documents.

Why Are Only the Child Nodes Inserted?

When appending a document fragment to a node, only the child nodes of the document fragment (but not the document fragment itself) are inserted. Wouldn't the parser check the owner document of these child nodes?

Answer: A document fragment should not be bound to a root node, since, by definition, a fragment could very well be just a list of nodes. The root node, if any, should be considered a single child. That is, you could for example take all the lines of an Invoice document, and add them into a ProviderOrder document, without taking the invoice itself. How do we create a document fragment without root? As the XSLT processor does, so that we can append it to other documents.

Why Do I Get DOMException when Setting Node Value?

I get the following error:

```
oracle.xml.parser.XMLDOMException: Node cannot be modified while trying to set
the value of a newly created node as below:
  String eName="Mynode";
  XMLNode aNode = new XMLNode(eName, Node.ELEMENT_NODE);
```

```
aNode.setNodeValue(eValue);
```

How do I create a node whose value I can set later on?

Answer: You will see that if you are creating an element node, its nodeValue is null and hence cannot be set.

How Can I Force the SAX Parser to Not Discard Characters Following Whitespace?

I receive the following error when reading the attached file using the SAX parser: if character data starts with a whitespace, `characters()` method discards characters that follow whitespace.

Is this a bug or can I force the parser to not discard those characters?

Answer: Use XMLParser.setPreserveWhitespace(true) to force the parser to not discard whitespace.

Frequently Asked Questions About Validation

What Are the Rules for Locating DTDs?

I have an XML string containing the following reference to a DTD, that is physically located in the directory where I start my program. The validating XML parser returns a message that this file cannot be found.

```
<!DOCTYPE xyz SYSTEM "xyz.dtd" >
```

What are the rules for locating DTDs on the disk?

Answer: Are you parsing an InputStream or a URL? If you are parsing an InputStream, the parser doesn't know where that InputStream came from so it cannot find the DTD in the “same directory as the current file”. The solution is to set`BaseURL()on DOMParser()` to give the parser the URL hint information to be able to derive the rest when it goes to get the DTD.

Can Multiple Threads Use a Single XSLProcessor/Stylesheet?

Can multiple threads use a single XSLProcessor / XSLStylesheet instance to perform concurrent transformations?

Answer: As long as you are processing multiple files with no more than one XSLProcessor / XSLStylesheet instance for each XML file you can do this simultaneously using threads. If you take a look at the `readme.html` file in the bin

directory, it describes ORAXSL which has a threads parameter for multithreaded processing.

Can I Use Document Clones in Multiple Threads?

Is it safe to use clones of a document in multiple threads? Is the public void `setParam(String,String)` throws `XSLExceptionmethod` of Class `oracle.xml.parser.v2.XSLStylesheet` supported? If no, is there another way to pass parameters at runtime to the XSLT processor?

Answer: If you are copying the global area set up by the constructor to another thread then it should work.

That method is supported since XML parser release 2.0.2.5.

Frequently Asked Questions About Character Sets

How Do I Parse iso-8859-1-encoded Documents with Special Characters?

I have some XML documents with ISO-8859-1 encoding. I am trying to parse these with the XML parser SAX API. In characters (`char[], int, int`), I would like to output the content in ISO-8859-1 (Latin1) too.

With `System.out.println()` it doesn't work correctly. German umlauts result in '?' in the output stream. What do I have to do to get the output in Latin1? The host system here is a Solaris™ Operating Environment 2.6.

Answer: You cannot use `System.out.println()`. You need to use an output stream which is encoding aware, for example, `OutputStreamWriter`.

You can construct an `outputstreamwriter` and use the `write(char[], int, int)` method to:

```
print.Ex:OutputStreamWriter out = new OutputStreamWriter(System.out, "8859_1");
/* Java enc string for ISO8859-1*/
```

How Do I Parse XML Stored in NCLOB with UTF-8 Encoding?

I'm having trouble with parsing XML stored in NCLOB column using UTF-8 encoding. Here is what I'm running:

- Windows NT 4.0 Server
- Oracle 8i (8.1.5)

- EEJDeveloper 3.0
- JDK 1.1.8
- Oracle XML Parser v2 (2.0.2.5?)

The following XML sample that I loaded into the database contains two UTF-8 multibyte characters:

```
<?xml version="1.0" encoding="UTF-8"?>
<G>
<A>GÂ,otingen, BrÃ ck_W</A>
</G>
```

The text is supposed to be:

G(0xc2, 0x82)otingen, Br(0xc3, 0xbc)ck_W

If I am not mistaken, both multibyte characters are valid UTF-8 encodings and they are defined in ISO-8859-1 as:

```
0xC2 LATIN CAPITAL LETTER A WITH CIRCUMFLEX
0xFC  LATIN SMALL LETTER U WITH DIAERESIS
```

I wrote a Java stored function that uses the default connection object to connect to the database, runs a Select query, gets the `OracleResultSet`, calls the `getCLOB()` method and calls the `getAsciiStream()` method on the CLOB object. Then it executes the following piece of code to get the XML into a DOM object:

```
DOMParser parser = new DOMParser();
parser.setPreserveWhitespace(true);
parser.parse(istr);
// istr getAsciiStreamXMLDocument xmldoc = parser.getDocument();
```

Before the stored function can do other tasks, this code throws an exception stating that the preceding XML contains invalid UTF-8 encoding.

- When I remove the first multibyte character (`0xc2, 0x82`) from the XML, it parses fine.
- When I do not remove this character, but connect through the JDBC Oracle thin driver (note that now I'm not running inside the RDBMS as stored function anymore) the XML is parsed with no problem and I can do what ever I want with the XML document.

I loaded the sample XML into the database using the thin JDBC driver. I tried two database configurations with `WE8ISO8859P1/WE8ISO8859P1` and `WE8ISO8859P1/UTF8` and both showed the same problem.

Answer: Yes, the character (`0xc2, 0x82`) is valid UTF-8. We suspect that the character is distorted when `getAsciiStream()` is called. Try to use `getUnicodeStream()` and `getBinaryStream()` instead of `getAsciiStream()`.

If this does not work, try to print out the characters to make sure that they are not distorted before they are sent to the parser in step: `parser.parse(istr)`

Is There Globalization Support Within XML?

I've got Japanese data stored in an `nvarchar2` field in the database. I have a dynamic SQL procedure that uses the PL/SQL web toolkit that enables me to access data using OAS and a browser. This procedure uses the XML parser to correctly format the result set in XML before returning it to the browser.

My problem is that the Japanese data is returned and displayed on the browser as upside down question marks. Is there anything I can do so that this data is correctly returned and displayed as Kanji?

Answer: Unfortunately, the Java and XML default character set is UTF-8 while I haven't heard of any UTF-8 operating systems nor people using it as in their database and people writing their web pages in UTF-8. All this means is that you have a character code conversion problem. The answer to your last question is yes. We do have both PL/SQL and Java XML parsers working in Japanese. Unfortunately, we cannot provide a simple solution that will fit in this space.

How Do I Parse a Document Containing Accented Characters?

This is my XML document:

Documento de Prueba de gestin de contenidos. Roberto P‰rez Lita

This is the way in which I parse the document:

```
DOMParser parser=new DOMParser();
parser.setPreserveWhitespace(true);
parser.setErrorStream(System.err);
parser.setValidationMode(false);
parser.showWarnings(true);
parser.parse ( new FileInputStream(new File("PruebaA3Ingles.xml")));
```

I get the following error:

```
XML-0231 : (Error) Encoding 'UTF-16' is not currently supported
```

I am using the XML Parser for Java version 2 and I am confused because the documentation says that the UTF-16 encoding is supported in this version of the Parser. Does anybody know how can I parse documents containing Spanish accents?

Answer: Oracle just uploaded a new release of the version 2 parser. It should support UTF-16. However, other utilities still have some problems with UTF-16 encoding.

How Do I Store Accented Characters in an XML Document?

I need to store accented characters in my XML documents. If I manually add an accented character, for example, an é, to my XML file and then attempt to parse the XML doc with the XML Parser for Java, the parser throws the following exception:

```
'Invalid UTF-8 encoding'
```

Here's the encoding declaration in my XML header:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Also, if I specify UTF-16 as the default encoding the parser states that UTF-16 is not currently supported. From within my Java program if I define a Java string object as follows:

```
String name = "éééé";
```

and programmatically generate an XML document and save it to file then the é character is correctly written out to file. Can you tell me how I can successfully read in character data consisting of accented characters? I know that I can read in accented characters once I represent them in their hex or decimal format within the XML document, for example:

```
&#xe9;
```

but I'd prefer not to do this.

Answer: You need to set the encoding based on the character set you were using when you created the XML file - I ran into this problem and solved it by setting the encoding to ISO-8859-1 (Western European ASCII) - you may need to use something different depending on the tool or operating system you are using.

If you explicitly set the encoding to UTF-8 (or do not specify it at all), the parser interprets your accented character (which has an ASCII value > 127) as the first byte of a UTF-8 multibyte sequence. If the subsequent bytes do not form a valid UTF-8 sequence, you get this error.

This error just means that your editor is not saving the file with UTF-8 encoding. For example, it might be saving it with ISO-8859-1 encoding. Remember that the encoding is a particular scheme used to write the Unicode character number representation to disk. Just adding the string to the top of the document like:

```
<?xml version="1.0" encoding="UTF-8"?>
```

does not cause your editor to write out the bytes representing the file to disk using UTF-8 encoding. I believe Notepad uses UTF-8, so you might try that.

Frequently Asked Questions: Adding an XML Document as a Child

How Do I Add an XML Document as a Child to Another Element?

I am trying to add an XML document as a child to an existing element. Here's an example:

```
import org.w3c.dom.*;
import java.util.*;
import java.io.*;
import java.net.*;
import oracle.xml.parser.v2.*;
public class ggg {public static void main (String [] args) throws Exception
 {
new ggg().doWork();;
public void doWork() throws Exception {XMLDocument doc1 = new XMLDocument();
Element root1=doc1.createElement("root1");
XMLDocument doc2= new XMLDocument();Element root2=doc2.createElement("root2");
root1.appendChild(root2);
doc1.print(System.out);};};
```

This reports:

```
D:\Temp\Oracle\sample>c:\jdk1.2.2\bin\javac -classpath
D:\Temp\Oracle\lib\xmlparserv2.jar;.
ggg.javaD:\Temp\Oracle\sample>c:\jdk1.2.2\bin\java -classpath
D:\Temp\Oracle\lib\xmlparserv2.jar;. gggException in thread "main"
java.lang.NullPointerException        at
oracle.xml.parser.v2.XMLDOMException.(XMLDOMException.java:67)        at
```

```
oracle.xml.parser.v2.XMLNode.checkDocument(XMLNode.java:919)        at
oracle.xml.parser.v2.XMLNode.appendChild(XMLNode.java, Compiled Code)        at
oracle.xml.parser.v2.XMLNode.appendChild(XMLNode.java:494)        at
ggg.doWork(ggg.java:20)        at ggg.main(ggg.java:12)
```

Answer 1: The following works for me:

```
DocumentFragment rootNode = new XMLDocumentFragment(); DOMParser d  = new
DOMParser(); d.parse("http://.../pfgrfff.xml");
Document doc = d.getDocument();
Element e = doc.getDocumentElement();
// Important to remove it from the first doc
// before adding it to the other doc. doc.removeChild(e);
rootNode.appendChild(e);
```

You need to use the `DocumentFragment` class to do this as a document cannot have more than one root.

Answer 2: Actually, isn't this specifically a problem with appending a node created in another document, since all nodes contain a reference to the document they are created in. While `DocumentFragment` solves this, it isn't a more than one root problem, is it? Is there a quick or easy way to convert a `com.w3c.dom.Document` to `org.w3c.dom.DocumentFragment`?

How Do I Add an XML Document Fragment as a Child to an XML Document?

I have this piece of code:

```
XSLStylesheet XSLProcessorStylesheet = new XSLStylesheet(XSLProcessorDoc,
XSLProcessorURL);
XSLStylesheet XSLRendererStylesheet = new XSLStylesheet(XSLRendererDoc,
XSLRendererURL);
XSLProcessor processor = new XSLProcessor();
// configure the processorprocessor.showWarnings(true);
processor.setErrorStream(System.err);
XMLDocumentFragment processedXML = processor.processXSL(XSLProcessorStylesheet,
XMLInputDoc);
XMLDocumentFragment renderedXML = processor.processXSL(XSLRendererStylesheet,
processedXML);
Document resultXML = new XMLDocument();
resultXML.appendChild(renderedXML);
```

The last line causes an exception in thread "main" `oracle.xml.parser.v2`.

```
XMLDOMException: Node of this type cannot be added.
```

Do I have to create a root element every time, even if I know that the resulting document fragment is a well formed XML document having only one root element?

Answer: It happens, as you have guessed, because a fragment can have more than one root element (for lack of a better term). In order to work around this, use the node functions to extract the one root element from your fragment and cast it into an

Frequently Asked General Questions About XML Parser

Why Do I Get an Error on Installing the XML Parser?

I get an error message when I try installing the XML parser:

```
loadjava -user username/manager -r -v xmlparserv2.jar
Error:
Exception in thread "main" java.lang.NoClassDefFounderr:
oracle/jdbc/driver/OracleDriver at oracle.aurora.server.tools.
```

Answer: This is a failure to find the JDBC `classes111.zip` in your CLASSPATH. The `loadjava` utility connects to the database to load your classes using the JDBC driver.

I checked 'loadjava' and the path to classes111.zip is

```
<ORACLE_HOME>/jdbc/lib/classes111.zip
```

In version 8.1.6, `classes111.zip` resides in:

```
<ORACLE_HOME/jdbc/admin
```

How Do I Remove the XML Parser from the Database?

How do I uninstall a version of the XML Parser and install a newer version? I know that there is something like `dropjava`, but still there are other packages which are loaded into the schema. I want to clean out the earlier version and install the new version in a clean manner.

Answer: You'll need to write SQL based on the `USER_OBJECTS` table where:

```
SELECT 'drop java class '''&#0124; &#0124;
dbms_java.longname(object_name)&#0124; &#0124;''';
from user_objects where

OBJECT_TYPE = 'JAVA CLASS'and DBMS_JAVA.LONGNAME(OBJECT_NAME)     LIKE
```

```
'oracle/xml/parser/%'
```

This will return a set of `DROP JAVA CLASS` commands which you can capture in a file using the SQL*Plus command `SPOOL somefilenamecommand`.

Then, run that spool file as a SQL script and all the right classes will be dropped.

What Does an XML Parser Do?

Answer: The parser accepts any XML document giving you a tree-based API (DOM) to access or modify the document's elements and attributes. It also includes an event API (SAX) that provides a listener to be registered, and report specific elements or attributes and other document events.

How Do I Convert XML Files into HTML Files?

Answer: You need to create an XSL stylesheet to render your XML into HTML. You can start with an HTML document in your desired format and populated with dummy data. Then you can replace this data with the XSLT commands that will populate the HTML with data from the XML document completing your stylesheet.

Does the XML Parser Validate Against XML Schema?

Does the XML Parser version 2 validate against an XML Schema?

Answer: Yes.

How Do I Include Binary Data in an XML Document?

How do I include binary data in an XML document?

Answer: There is no way to directly include binary data within the document; however, there are two ways to work around this:

- Binary data can be referenced as an external unparsed entity that resides in a different file.
- Binary data can be uuencoded (meaning converted into ASCII data) and be included in a CDATA section. The limitation on the encoding technique is to ensure that it only produces legal characters for the CDATA section.

What Is XML Schema?

Answer: XML Schema is a W3C XML standards effort to bring the concept of data types to XML documents and in the process replace the syntax of DTDs to one based on XML. For more details, visit the following Web sites:

`http://www.w3.org/TR/xmlschema-1/`

`http://www.w3.org/TR/xmlschema-2/`

XML Schema is supported in Oracle9*i* and higher.

Does Oracle Participate in Defining the XML/XSL Standard?

Answer: Oracle has representatives participating actively in the following 3C Working Groups related to XML / XSL: XML Schema, XML Query, XSL, XLink / XPointer, XML Infoset, DOM, and XML Core.

How Do I Find XDK Version Numbers?

How do I determine the version number of the XDK toolkit that I downloaded?

Answer: You can find out the full version number by looking at the `readme.html` file included in the archive and linked to the Release Notes page.

Are Namespace and Schema Supported?

Answer: The current XML parsers support Namespaces. Schema support is provided in Oracle9*i* and higher.

Can I Use JDK 1.1.x with XML Parser for Java v2?

Can I use JDK 1.1.x with XML Parser v2 for Java?

Answer: Version 2 of the XML Parser for Java has nothing to do with Java2. It is simply a designation that indicates that it is not backward compatible with the version 1 parser and that it includes XSLT support. Version 2 of the parser will work fine with JDK 1.1.x.

How Do I Sort the Result Within the Page?

I have a set of 100 records, and I am showing 10 at a time. On each column name I have made a link. When that link is clicked, I want to sort the data in the page alone, based on that column. How do I go about this?

Answer: If you are writing for IE5 alone and receiving XML data, you could just use Microsoft's XSL to sort data in a page. If you are writing for another browser and the browser is getting the data as HTML, then you have to have a sort parameter in XSQL script and use it in `ORDER BY` clause. Just pass it along with the skip-rows parameter.

Do I Need Oracle9*i* to Run XML Parser for Java?

Answer: XML Parser for Java can be used with any of the supported version JavaVMs. The only difference with Oracle9*i* is that you can load it into the database and use JServer, which is an internal JVM. For other database versions or servers, you simply run it in an external JVM and as necessary connect to a database through JDBC.

Can I Dynamically Set the Encoding in an XML File?

Answer: No, you need to include the proper encoding declaration in your document according to the specification. You cannot use `setEncoding()` to set the encoding for you input document. `SetEncoding()` is used with `oracle.xml.parser.v2.XMLDocument` to set the correct encoding for the printing.

How Do I Parse a String?

Answer: We do not currently have any method that can directly parse an XML document contained within a string. You would need to convert the string into an InputStream or InputSource before parsing. An easy way is to create a ByteArrayInputStream using the bytes in the string.

How Do I Display an XML Document?

Answer: If you are using IE5 as your browser you can display the XML document directly. Otherwise, you can use the Oracle XSLT Processor version 2 to create the HTML document using an XSL Stylesheet. The Oracle XML Transviewer bean also enables you to view your XML document.

How Do I Use System.out.println() and Special Characters?

Answer: You can't use `System.out.println()`. You need to use an output stream which is encoding aware (for example, OutputStreamWriter). You can

construct an OutputStreamWriter and use the `write(char[], int, int)` method to print.

```
/* Example */
OutputStreamWriter out = new OutputStreamWriter
(System.out, "8859_1");
/* Java enc string for ISO8859-1*/
```

How Do I Insert Characters <, >, =, ', ", and & in XML Documents?

How do I insert these characters in the XML documents: greater than (>), less than (<), apostrophe, double quotes, or equals (=)?

Answer: You need to use the *entity references* `&eq;` for equals (=), `>` for greater than (>), and `<` for less than (<). Use ' for an apostrophe or single quote. Use " for straight double quotes. Use & for ampersand.

How Do I Use Special Characters in the Tags?

I have a tag in XML `<COMPANYNAME>`

When we try to use `A&B`, the parser gives an error with invalid character. How do we use special characters when parsing companyname tag? We are using the Oracle XML Parser for C.

Answer: You can use special characters as part of XML name. For example: `<A&B>abc</A&B>`

If this is the case, using name entity doesn't solve the problem. According to XML 1.0 spec, `NameChar` and `Name` are defined as follows:

```
NameChar ::= Letter | Digit | '.' | '-' | '_' | ':' | CombiningChar |Extender
Name     ::= (Letter | '_' | ':') (NameChar)*
```

To answer your question, special characters such as &, $, and #, and so on are not allowed to be used as `NameChar`. Hence, if you are creating an XML document from scratch, you can use a workaround by using only valid `NameChars`. For example, `<A_B>, <AB>, <A_AND_B>` and so on.

They are still readable.

If you are generating XML from external data sources such as database tables, then this is a problem which XML 1.0 does not address.

In Oracle, the new type, `XMLType`, will help address this problem by offering a function which maps SQL names to XML names. This will address this problem at

the application level. The SQL to XML name mapping function will escape invalid XML NameChar in the format of _XHHHH_ where HHHH is a Unicode value of the invalid character. For example, table name V$SESSION will be mapped to XML name V_X0024_SESSION.

Finally, escaping invalid characters is a workaround to give people a way to serialize names so that they can reload them somewhere else.

How Do I Parse XML from Data of Type String?

Answer: Check out the following example:

```
/* xmlDoc is a String of xml */
byte aByteArr [] = xmlDoc.getBytes();
ByteArrayInputStream bais = new ByteArrayInputStream (aByteArr, 0,
aByteArr.length);
domParser.parse(bais);
```

How Do I Extract Data from an XML Document into a String?

Answer: Here is an example to do that:

```
XMLDocument Your Document;
/* Parse and Make Mods */
:
StringWriter sw = new StringWriter();
PrintWriter  pw = new PrintWriter(sw);
YourDocument.print(pw);
String YourDocInString = sw.toString();
```

Is Disabling Output Escaping Supported?

Answer: Yes, since release 2.022, the XML Parser for Java provides an option to xsl:text to disable output escaping.

Can I Delimit Multiple XML Documents with a Special Character?

We need to be able to read and separate several XML documents as a single string. One solution would be to delimit these documents using some program-generated special character that we know for sure can never occur inside an XML document. The individual documents can then be easily tokenized and extracted or parsed as required.

Has any one else done this before? Any suggestions for what character can be used as the delimiter? For instance can characters in the range #x0-#x8 ever occur inside an XML document?

Answer: As far as legality is concerned, and if you limit it to 8-bit, then #x0-#x8; #xB, #xC, #xE, and #xF are not legal. However, this assumes that you preprocess the doc and do not depend upon exceptions as not all parsers reject all illegal characters.

How Do I Use Entity References with the XML Parser for Java?

The XML parser for Java does not expand entity references, such as &[whatever]. Instead, all values are null. How can I fix this?

Answer: You probably have a simple error defining or using your entities, since we have a number of regression tests that handle entity references fine. A simple example is:]> Alpha, then &status.

Can I Divide and Store an XML Document Without a DDL Insert?

We would like to break apart an arbitrary XML document and store it in the database without creating a DDL to insert. Is this possible?

Answer: In Oracle8*i* release 8.1.6 and higher, Oracle Text can do this.

In Querying, Can I Perform Hierarchical Searches Across XML Documents?

Answer: No this is not possible. Either the schema must already exist or and XSL stylesheet to create the DDL from the XML must exist.

How Do I Merge XML Documents?

Answer: This is not possible with the current DOM1 specification. The DOM2 specification may address this.

As a workaround, you can use a DOM approach or an XSLT-based approach to accomplish this. If you use DOM, then you'll have to remove the node from one document before you append it into the other document to avoid ownership errors.

Here is an example of the XSL-based approach. Assume your two XML source files are:

demo1.xml

```
<messages>
```

```
  <msg>
    <key>AAA</key>
    <num>01001</num>
  </msg>
  <msg>
    <key>BBB</key>
    <num>01011</num>
  </msg>
</messages>
```

demo2.xml

```
<messages>
  <msg>
    <key>AAA</key>
    <text>This is a Message</text>
  </msg>
  <msg>
    <key>BBB</key>
    <text>This is another Message</text>
  </msg>
</messages>
```

Here is a stylesheet that joins `demo1.xml` to `demo2.xml` based on matching the `<key>` values.

demomerge.xsl

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output indent="yes"/>
<xsl:variable name="doc2" select="document('demo2.xml')"/>
  <xsl:template match="@*|node()">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
  </xsl:template>
<xsl:template match="msg">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()"/>
      <text><xsl:value-of select="$doc2/messages/msg[key=current()/key]/text"/>
</text>
    </xsl:copy>
</xsl:template>
</xsl:stylesheet>
```

If you use the command line `oraxsl` to test this, you would enter:

```
$ oraxsl demo1.xml demomerge.xsl
```

Then, you will get the following merged result:

```
<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>
```

This is obviously not as efficient for larger files as an equivalent database join between two tables, but this illustrates the technique if you have only XML files to work with.

How Do I Find the Value of a Tag?

I am using SAX to parse an XML document. How I can get the value of a particular tag? For example, in Java, how do I get the value for `title`? I know there are `startElement`, `endElement`, and `characters` methods.

Answer: During a SAX parse the value of an element will be the concatenation of the characters reported from after `startElement` to before the corresponding `endElement` is called.

How Do I Grant the JAVASYSPRIV Role to a User?

We are using Oracle XML Parser for Java on Windows NT 4.0. When we are parsing an XML document with an external DTD we get the following error:

```
<!DOCTYPE listsamplereceipt SYSTEM
"file:/E:/ORACLE/utl_file_dir/dadm/ae.dtd">
java.lang.SecurityExceptionat
oracle.aurora.rdbms.SecurityManagerImpl.checkFile(SecurityManagerImpl.java)at
oracle.aurora.rdbms.SecurityManagerImpl.checkRead(SecurityManagerImpl.java)at
java.io.FileInputStream.<init>(FileInputStream.java)at
java.io.FileInputStream.<init>(FileInputStream.java)at
sun.net.www.MimeTable.load(MimeTable.java)at
sun.net.www.MimeTable.<init>(MimeTable.java)at
sun.net.www.MimeTable.getDefaultTable(MimeTable.java)at
```

```
sun.net.www.protocol.file.FileURLConnection.connect(FileURLConnection.java)at
sun.net.www.protocol.file.FileURLConnection.getInputStream(FileURLConnection.
java)at
java.net.URL.openStream(URL.java)at
oracle.xml.parser.v2.XMLReader.openURL(XMLReader.java:2313)at
oracle.xml.parser.v2.XMLReader.pushXMLReader(XMLReader.java:176)at
...
```

What is causing this?

Answer: Grant the JAVASYSPRIV role to your user running this code to allow it to open the external file or URL.

How Do I Include an External XML File in Another XML File?

I am trying to include an external XML file in another XML file. Do the XML Parser for Java version 1 and version 2 support external parsed entities?

Answer: IE 5.0 will parse an XML file and show the parsed output. Just load the file as you would an HTML page.

The following works, both browsing it in IE5 as well as parsing it with the XML Parser for Java version 2. Even though I'm sure it works fine in the XML Parser for Java version 1, you should be using the latest parser version as it is faster than version 1.

```
File: a.xml
<?xml version="1.0" ?>
<!DOCTYPE a [<!ENTITY b SYSTEM "b.xml">]>
 <a>&b;</a>

 File: b.xml
 <ok/>
```

When I browse and parse a.xml I get the following:

```
<a>
  <ok/>
</a>
```

Does the Parser Come with a Utility to View the Parsed Output?

We are using the XML Parser for Java version 1.0, because that is what is shipped to the customers with release 10.7 and 11.0 of our application. Can you refer me to this, or some other sample code to do this.

Shouldn't file `b.xml` be in the format:

```
<?xml version="1.0" ?>
<b>
  <ok/>
</b>
```

Does the Oracle XML Parser come with a utility to parse an XML file and see the parsed output?

Answer: Not strictly. The parsed external entity only needs to be a well-formed fragment. The following program (with `xmlparser.jar` from version 1) in your `CLASSPATH` shows parsing and printing the parsed document. It's parsing here from a string but the mechanism would be no different for parsing from a file, given its URL.

```
import oracle.xml.parser.*;
import java.io.*;
import java.net.*;
import org.w3c.dom.*;
import org.xml.sax.*;
/*
** Simple Example of Parsing an XML File from a String
** and, if successful, printing the results.
**
** Usage: java ParseXMLFromString <hello><world/></hello>
*/
public class ParseXMLFromString {
  public static void main( String[] arg ) throws IOException, SAXException {
    String theStringToParse =
       "<?xml version='1.0'?>"+
       "<hello>"+
       "  <world/>"+
       "</hello>";
    XMLDocument theXMLDoc = parseString( theStringToParse );
    // Print the document out to standard out
    theXMLDoc.print(System.out);
  }
  public static XMLDocument parseString( String xmlString ) throws
   IOException, SAXException {
   XMLDocument theXMLDoc     = null;
    // Create an oracle.xml.parser.v2.DOMParser to parse the document.
    XMLParser theParser = new XMLParser();
    // Open an input stream on the string
    ByteArrayInputStream theStream =
```

```
            new ByteArrayInputStream( xmlString.getBytes() );
        // Set the parser to work in non-Validating mode
        theParser.setValidationMode(DTD_validation);
        try {
          // Parse the document from the InputStream
          theParser.parse( theStream );
          // Get the parsed XML Document from the parser
          theXMLDoc = theParser.getDocument();
        }
        catch (SAXParseException s) {
          System.out.println(xmlError(s));
          throw s;
        }
        return theXMLDoc;
      }
      private static String xmlError(SAXParseException s) {
         int lineNum = s.getLineNumber();
         int  colNum = s.getColumnNumber();
         String file = s.getSystemId();
         String  err = s.getMessage();
         return "XML parse error in file " + file +
                "\n" + "at line " + lineNum + ", character " + colNum +
                "\n" + err;
      }
    }
```

From Where Can I Download OraXSL, the Parser's Command Line Interface?

From where I can download `oracle.xml.parser.v2.OraXSL`?

Answer: It's part of our integrated XML Parser for Java version 2 release. Our XML Parser, DOM, XPath implementation, and XSLT engine are nicely integrated into a single cooperating package. To download it, please refer to the following Web site:

`http://otn.oracle.com/tech/xml/xdk_java/`

Does Oracle Support Hierarchical Mapping?

We are interested in using the Oracle database primarily to store XML. We would like to parse incoming XML documents and store data and tags in the database. We are concerned about the following two aspects of XML in Oracle:

First, the relational mapping of parsed XML data. We prefer hierarchical storage of parsed XML data. Is this a valid concern? Will XMLType in Oracle9*i* address this concern?

Second, a lack of an ambiguous content mode in the Oracle Parser for Java is limiting to our business. Are there plans to add an ambiguous content mode to the Oracle Parser for Java?

Answer: Many customers initially have this concern. It depends on what kind of XML data you are storing. If you are storing XML datagrams that are really just encoding of relational information (for example, a purchase order), then you will get much better performance and much better query flexibility (in SQL) to store the data contained in the XML documents in relational tables, then reproduce on-demand an XML format when any particular data needs to be extracted.

If you are storing documents that are mixed-content, like legal proceedings, chapters of a book, reference manuals, and so on, then storing the documents in chunks and searching them using Oracle Text's XML search capabilities is the best bet.

The book, *Building Oracle XML Applications*, by Steve Muench, covers both of these storage and searching techniques with lots of examples.

> **See Also:** ?For more information on using Oracle Text and XML, see:
>
> - *Oracle Text Reference*
> - *Oracle Text Application Developer's Guide*
> - `http://otn.oracle.com/products/text`

For the second point, the Oracle XML Parser implements all the XML 1.0 standard, and the XML 1.0 standard requires XML documents to have unambiguous content models. Therefore, there is no way a compliant XML 1.0 parser can implement ambiguous content models.

> **See Also:**
> `http://www.xml.com/axml/target.html#determinism`

What Good Books for XML/XSL Can You Recommend?

Can any one suggest good books for learning about XML and XSL?

Answer: There are many excellent articles, white papers, and books that describe all facets of XML technology. Many of these are available on the World Wide Web. The following are some of the most useful resources we have found:

- XML, Java, and the Future of the Web by Jon Bosak, Sun Microsystems `http://metalab.unc.edu/pub/sun-info/standards/xml/why/xmlapps.htm`
- XML for the Absolute Beginner by Mark Johnson, JavaWorld `http://www.javaworld.com/jw-04-1999/jw-04-xml_p.html`
- XML And Databases by Ronald Bourret, Technical University of Darmstadt `http://www.informatik.tu-darmstadt.de/DVS1/staff/bourret/XML/`
- XMLAndDatabases.htm and the XML Specifications by the World Wide Web Consortium (W3C) `http://www.w3.org/XML/`
- XML.com, a broad collection of XML resources and commentary `http://www.xml.com/`
- Annotated XML Specification by Tim Bray, XML.com `http://www.xml.com/axml/testaxml.htm`
- The XML FAQ by the W3C XML Special Interest Group (the industry clearing house for XML DTDs that allow companies to exchange XML data) `http://www.ucc.ie/xml/ XML.org`
- `http://xml.org/`
- xDev (the DataChannel XML Developer pages) `http://xdev.datachannel.com/`

Are There XML Developer Kits for the HP/UX Platform?

Answer: HP-UX ports for our C/C++ Parser as well as our C++ Class Generator are available. Look for an announcement on `http://technet.oracle.com`

How Do I Compress Large Volumes of XML Documents?

Can we compress XML documents when saving them to the database as a CLOB? If they are compressed, what is the implication of using Oracle Text against the documents? We have large XML documents that range up to 1 MB and they need to be minimized.

The main requirement is to save cost in terms of disk storage as the XML documents stored are history information (more of a datawarehouse environment). We could save a lot of disk space if we could compress the documents before storage. The searching capability is only secondary, but a big plus.

Answer: The XDK for Java supports a compression mechanism in Oracle9*i*. It supports streaming compression and uncompression. The compression is achieved by removing the markup in the XML Document. The initial version does not support searching the compressed data. This is planned for a future release.

If you want to store and search your XML docs, Oracle Text can handle this. I am sure that the size of individual document is not a problem for Oracle Text.

If you want to compress the 1 MB docs for saving disk space and costs, Oracle Text will not be able to automatically handle a compressed XML document.

Try looking at XMLZip:

`http://www.xmls.com/resources/xmlzip.xml?id=resources_xmlzip`

My only concern would be the performance hit to do the uncompression. If you are just worried about transmitting the XML from client to server or vice versa, then HTTP compression could be easier.

How Do I Generate an XML Document Based on Two Tables?

I would like to generate an XML document based on two tables with a master detail relationship. Suppose I have two tables:

- PARENT with columns: ID and PARENT_NAME (Key = ID)
- CHILD with columns: PARENT_ID, CHILD_ID, CHILD_NAME (Key = PARENT_ID + CHILD_ID)

There is a master detail relationship between `PARENT` and `CHILD`. How can I generate a document that looks like this?

```
<?xml version = '1.0'?>
  <ROWSET>
     <ROW num="1">
       <parent_name>Bill</parent_name>
         <child_name>Child 1 of 2</child_name>
         <child_name>Child 2 of 2</child_name>
      </ROW>
      <ROW num="2">
       <parent_name>Larry</parent_name>
         <child_name>Only one child</child_name>
      </ROW>
  </ROWSET>
```

Answer: You should use an object view to generate an XML document from a master-detail structure. In your case, use the following code:

```
create type child_type is object
(child_name <data type child_name>) ;
/
create type child_type_nst
is table of child_type ;
/

create view parent_child
as
select p.parent_name
, cast
  ( multiset
    ( select c.child_name
      from   child c
      where  c.parent_id = p.id
    ) as child_type_nst
  ) child_type
from parent p
/
```

A `SELECT * FROM parent_child`, processed by an SQL to XML utility would generate a valid XML document for your parent child relationship. The structure would not look like the one you have presented, though. It would look like this:

```
<?xml version = '1.0'?>
<ROWSET>
   <ROW num="1">
      <PARENT_NAME>Bill</PARENT_NAME>
      <CHILD_TYPE>
         <CHILD_TYPE_ITEM>
            <CHILD_NAME>Child 1 of 2</CHILD_NAME>
         </CHILD_TYPE_ITEM>
         <CHILD_TYPE_ITEM>
            <CHILD_NAME>Child 2 of 2</CHILD_NAME>
         </CHILD_TYPE_ITEM>
      </CHILD_TYPE>
  </ROW>
   <ROW num="2">
      <PARENT_NAME>Larry</PARENT_NAME>
      <CHILD_TYPE>
         <CHILD_TYPE_ITEM>
            <CHILD_NAME>Only one child</CHILD_NAME>
```

```
            </CHILD_TYPE_ITEM>
        </CHILD_TYPE>
    </ROW>
</ROWSET>
```

5

XSLT Processor for Java

This chapter contains the following sections:

Using XML Parser for Java: XSLT Processor

The XSLT processor operates on two inputs: the XML document to transform, and the XSLT stylesheet that is used to apply transformations on the XML. Each of these two can actually be multiple inputs. One stylesheet can be used to transform multiple XML inputs. Multiple stylesheets can be mapped to a single XML input.

To implement the XSLT Processor in the XML Parser for Java use XSLProcessor class.

Figure 5–1 shows the overall process used by class `XSLProcessor`. Here are the steps:

- Create an XSLProcessor object and then use methods from the following list in your Java code. Some of the available methods are:
  - `removeParam()` - remove parameter
  - `resetParam()` - remove all parameters
  - `setParam()` - set parameters for the transformation
  - `setBaseURL()` - set a base URL for any relative references in the stylesheet
  - `setEntityResolver()` - set an entity resolver for any relative references in the stylesheet
  - `setLocale` - set locale for error reporting
- Use one of the following input parameters to the function `XSLProcessor.newXSLStylesheet()` to create a stylesheet object:
  - `java.io.Reader`
  - `java.io.InputStream`
  - `XMLDocument`
  - `java.net.URL`

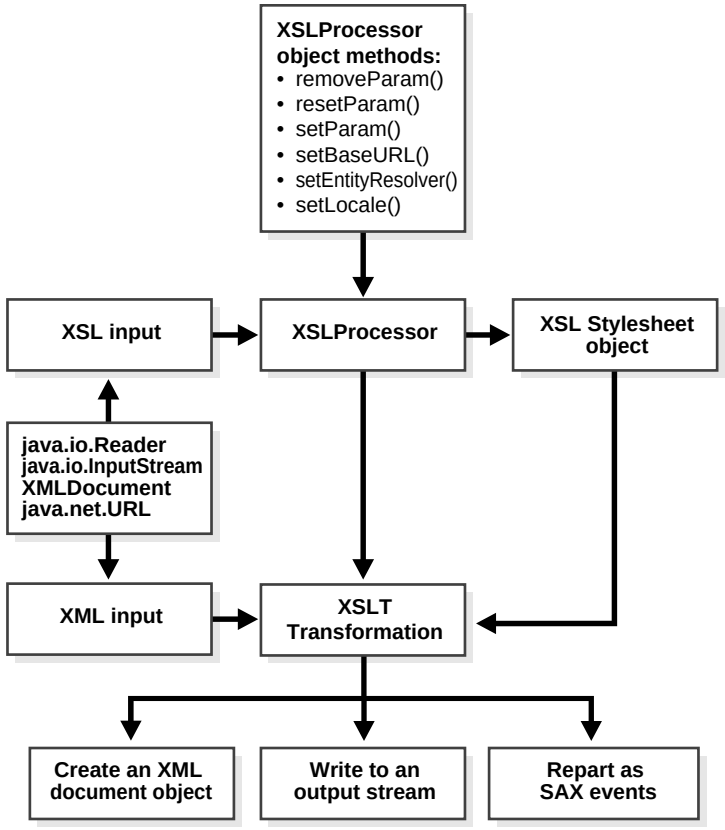
  This creates a stylesheet object which is thread-safe and can be used in multiple XSL Processors.
- Use one of the input parameters on the XML input.
- Your XML inputs and the stylesheet object are input (each using one of the input parameters listed above) to the XSL Processor:

  ```
  XSLProcessor.processXSL(xslstylesheet, xml instance)
  ```

The results can be one of the following:

- create an XML document object
- write to an output stream
- report as SAX events

Figure 5–1 Using XSL Processor for Java



XSLT Processor for Java Example

This example uses one XML document and one XSTT stylesheet as inputs.

```
public class XSLSample
```

```
{
   public static void main(String args[]) throws Exception
   {
      if (args.length < 2)
      {
         System.err.println("Usage: java XSLSample xslFile xmlFile.");
         System.exit(1);
      }

      // Create a new XSLProcessor.
      XSLProcessor processor = new XSLProcessor();

      // Register a base URL to resolve relative references
      // processor.setBaseURL(baseURL);

      // Or register an org.xml.sax.EntityResolver to resolve
      // relative references
      // processor.setEntityResolver(myEntityResolver);

      // Register an error log
      // processor.setErrorStream(new FileOutputStream("error.log"));

      // Set any global paramters to the processor
      // processor.setParam(namespace, param1, value1);
      // processor.setParam(namespace, param2, value2);

      // resetParam is for multiple XML documents with different parameters

      String xslFile = args[0];
      String xmlFile = args[1];

      // Create a XSLStylesheet
      //  The stylesheet can be created using one of following inputs:
      //
      // XMLDocument xslInput = /* using DOMParser; see below in this code */
      // URL         xslInput = new URL(xslFile);
      // Reader      xslInput = new FileReader(xslFile);

      InputStream xslInput = new FileInputStream(xslFile);

      XSLStylesheet stylesheet = processor.newXSLStylesheet(xslInput);

      // Prepare the XML instance document
      //   The XML instance can be given to the processor in one of
      // following ways:
```

```
        //
        // URL         xmlInput = new URL(xmlFile);
        // Reader      xmlInput = new FileReader(xmlFile);
        // InputStream xmlInput = new FileInputStream(xmlFile);
        // Or using DOMParser

        DOMParser parser = new DOMParser();
        parser.retainCDATASection(false);
        parser.setPreserveWhitespace(true);
        parser.parse(xmlFile);
        XMLDocument xmlInput = parser.getDocument();

        // Transform the XML instance
        //   The result of the transformation can be one of the following:
        //
        // 1. Return a XMLDocumentFragment
        // 2. Print the results to a OutputStream
        // 3. Report SAX Events to a ContentHandler

        // 1. Return a XMLDocumentFragment
        XMLDocumentFragment result;
        result = processor.processXSL(stylesheet, xmlInput);

        // Print the result to System.out
        result.print(System.out);

        // 2. Print the results to a OutputStream
        // processor.processXSL(stylesheet, xmlInput, System.out);

        // 3. Report SAX Events to a ContentHandler
        // ContentHandler cntHandler = new MyContentHandler();
        // processor.processXSL(stylesheet, xmlInput, cntHandler);

    }
}
```

See Also: .See "SAX: Event-Based API" on page 4-8

XSLT Processor for Java: Command-Line Interface, oraxsl

oraxsl - Oracle XSL processor

`oraxsl` is a command-line interface used to apply a stylesheet on multiple XML documents. It accepts a number of command-line options that dictate how it should behave.

To use `oraxsl` ensure the following:

- Your CLASSPATH environment variable is set to point to the xmlparserv2.jar file that comes with Oracle XML V2 parser for Java.
- Your PATH environment variable can find the java interpreter that comes with JDK 1.1.x or JDK 1.2.

Use the following syntax to invoke `oraxsl`:

```
oraxsl options source stylesheet result
```

`oraxsl` expects to be given a stylesheet, an XML file to transform, and optionally, a result file. If no result file is specified, it outputs the transformed document to standard out. If multiple XML documents need to be transformed by a stylesheet, the -l or -d options in conjunction with the -s and -r options should be used instead. These and other options are described in Table 5–1.

Table 5–1 oraxsl: Command Line Options

Option	Purpose
-d *directory*	Directory with files to transform (the default behavior is to process all files in the directory). If only a certain subset of the files in that directory, for example, one file, need to be processed, this behavior must be changed by using -l and specifying just the files that need to be processed. You could also change the behavior by using the '-x' or '-i' option to select files based on their extension).
-debug	New - Debug mode (by default, debug mode is turned off)
-e *error_log*	A file to write errors to (specify a log file to write errors and warnings).
-h	Help mode (prints oraxsl invocation syntax)
-i *source_extension*	Extensions to include (used in conjunction with -d. Only files with the specified extension will be selected).

Table 5–1 oraxsl: Command Line Options (Cont.)

Option	Purpose
-l *xml_file_list*	List of files to transform (enables you to explicitly list the files to be processed).
-o *result_directory*	Directory to place results (this must be used in conjunction with the -r option).
-p *param_list*	List of Parameters.
-r *result_extension*	Extension to use for results (if -d or -l is specified, this option must be specified to specify the extension to be used for the results of the transformation. So, if one specifies the extension "out", an input document "foo" would get transformed to "foo.out". By default, the results are placed in the current directory. This is can be changed by using the -o option which enables you to specify a directory to hold the results).
-s *stylesheet*	Stylesheet to use (if -d or -l is specified, this option needs to be specified to specify the stylesheet to be used. The complete path must be specified).
-t *num_of_threads*	Number of threads to use for processing (using multiple threads could provide performance improvements when processing multiple documents).
-v	Verbose mode (some debugging information is printed and could help in tracing any problems that are encountered during processing)
-w	Show warnings (by default, warnings are turned off)
-x *source_extension*	Extensions to exclude (used in conjunction with -d. All files with the specified extension will not be selected).

XML Extension Functions for XSLT Processing

XML extension functions for XSLT processing allow users of XSLT processor to call any Java method from XSL expressions.

XSLT Processor Extension Functions: Introduction

Java extension functions should belong to the namespace that starts with the following:

```
http://www.oracle.com/XSL/Transform/java/
```

An extension function that belongs to the following namespace:

```
http://www.oracle.com/XSL/Transform/java/classname
```

refers to methods in class `classname`. For example, the following namespace:

```
http://www.oracle.com/XSL/Transform/java/java.lang.String
```

can be used to call `java.lang.String` methods from XSL expressions.

Static Versus Non-Static Methods

If the method is a non-static method of the class, then the first parameter will be used as the instance on which the method is invoked, and the rest of the parameters are passed on to the method.

If the extension function is a static method, then all the parameters of the extension function are passed on as parameters to the static function.

XML Parser for Java - XSL Example 1: Static function

The following XSL, static function example:

```
<xsl:stylesheet
xmlns:math="http://www.oracle.com/XSL/Transform/java/java.lang.Math">
  <xsl:template match="/">
  <xsl:value-of select="math:ceil('12.34')"/>
</xsl:template>
</xsl:stylesheet>
```

prints out '13'.

Note: The XSL class loader only knows about statically added JARs and paths in the CLASSPATH - those specified by `wrapper.classpath`. Files added dynamically using the repositories' keyword in Jserv are not visible to XSL processor.

Constructor Extension Function

The extension function 'new' creates a new instance of the class and acts as the constructor.

XML Parser for Java - XSL Example 2: Constructor Extension Function

The following constructor function example:

```
<xsl:stylesheet
xmlns:jstring="http://www.oracle.com/XSL/Transform/java/java.lang.String">
  <xsl:template match="/">
  <!-- creates a new java.lang.String and stores it in the variable str1 -->
  <xsl:variable name="str1" select="jstring:new('Hello World')"/>
  <xsl:value-of select="jstring:toUpperCase($str1)"/>
</xsl:template>
</xsl:stylesheet>
```

prints out 'HELLO WORLD'.

Return Value Extension Function

The result of an extension function can be of any type, including the five types defined in XSL:

- NodeList
- boolean
- String
- Number
- resulttree

They can be stored in variables or passed onto other extension functions.

If the result is of one of the five types defined in XSL, then the result can be returned as the result of an XSL expression.

XML Parser for Java XSL- XSL Example 3: Return Value Extension Function

Here is an XSL example illustrating the Return value extension function:

```
<!-- Declare extension function namespace -->
<xsl:stylesheet xmlns:parser =
"http://www.oracle.com/XSL/Transform/java/oracle.xml.parser.v2.DOMParser"
xmlns:document =
"http://www.oracle.com/XSL/Transform/java/oracle.xml.parser.v2.XMLDocument" >

<xsl:template match ="/"> <!-- Create a new instance of the parser, store it in
myparser variable -->
<xsl:variable name="myparser" select="parser:new()"/>
<!-- Call a non-static method of DOMParser. Since the method is anon-static
method, the first parameter is the instance on which themethod is called. This
is equivalent to $myparser.parse('test.xml') -->
```

```
<xsl:value-of select="parser:parse($myparser, 'test.xml')"/>
<!-- Get the document node of the XML Dom tree -->
<xsl:variable name="mydocument" select="parser:getDocument($myparser)"/>
<!-- Invoke getelementsbytagname on mydocument -->
<xsl:for-each select="document:getElementsByTagName($mydocument,'elementname')">
......
</xsl:for-each> </xsl:template>
</xsl:stylesheet>
```

Datatypes Extension Function

Overloading based on number of parameters and type is supported. Implicit type conversion is done between the five XSL types as defined in XSL.

Type conversion is done implicitly between (String, Number, Boolean, ResultTree) and from NodeSet to (String, Number, Boolean, ResultTree).

Overloading based on two types which can be implicitly converted to each other is not permitted.

XML Parser for Java - XSL Example 4: Datatype Extension Function

The following overloading will result in an error in XSL, since String and Number can be implicitly converted to each other:

- abc(int i){}
- abc(String s){}

Mapping between XSL type and Java type is done as following:

```
String -> java.lang.String
Number -> int, float, double
Boolean -> boolean
NodeSet -> XMLNodeList
ResultTree -> XMLDocumentFragment
```

Oracle XSLT Built-In Extensions: ora:node-set and ora:output

The following example illustrates both ora:node-set and ora:output in action.

If you enter:

```
$ oraxsl foo.xml slides.xsl toc.html
```

where "foo.xml" is any XML file, you get:

- A "toc.html" slide with a table of contents
- A "slide01.html" file with slide 1
- A "slide02.html" file with slide 2

```
<!--
    | Illustrate using ora:node-set and ora:output
    |
    | Both extensions depend on defining a namespace
    | with the uri of "http://www.oracle.com/XSL/Transform/java"
+-->
<xsl:stylesheet version="1.0"

xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

xmlns:ora="http://www.oracle.com/XSL/Transform/java">

<!-- <xsl:output> affects the primary result document -->
<xsl:output mode="html" indent="no"/>

<!--
     |   <ora:output> at the top-level enables all attributes
     |   that <xsl:output> enables, but you must provide the
     |   additional "name" attribute to assign a name to
     |   these output settings to be used later.
+-->
<ora:output name="myOutput" mode="html" indent="no"/>
<!--
     | This top-level variable is a result-tree fragment
+-->
<xsl:variable name="fragment">
   <slides>
       <slide>
          <title>First Slide</title>
             <bullet>Point One</bullet>
             <bullet>Point Two</bullet>
             <bullet>Point Three</bullet>
       </slide>
       <slide>
          <title>Second Slide</title>
              <bullet>Point One</bullet>
              <bullet>Point Two</bullet>
              <bullet>Point Three</bullet>
       </slide>
   </slides>
```

```
</xsl:variable>
<xsl:template match="/">
<!--    | We cannot "de-reference" a result-tree-fragment to
        | navigate into it with an XPath expression. However, using
        | the ora:node-set() built-in extension function, you can
        | "cast" a result-tree fragment to a node-set which *can*
        | then be navigated using XPath. Since we'll use the node-set
        | of <slides> twice below, we save the node-set in a variable.
+-->
<xsl:variable name="slides" select="ora:node-set($fragment)"/>
<!--
    | This <html> page will go to the primary result document.
    | It is a "table of contents" for the slide show, with
    | links to each slide. The "slides" will each be generated
    | into *secondary* result documents, each slide having
    | a file name of "slideNN.html" where NN is the two-digit
    | slide number
+-->
<html>
   <body>
      <h1>List of All Slides</h1>
<xsl:apply-templates select="$slides" mode="toc"/>
   </body>
</html>
<!--
     | Now go apply-templates to format each slide
+-->
<xsl:apply-templates select="$slides"/>
</xsl:template>
<!-- In 'toc' mode, generate a link to each slide we match -->
<xsl:template match="slide" mode="toc">
     <a href="slide{format-number(position(),'00')}.html">
<xsl:value-of select="title"/>
     </a><br/>
</xsl:template>
<!--
      | For each slide matched, send the output for the current
      | <slide> to a file named "slideNN.html". Use the named
      | output style defined above called "myOutput".
<xsl:template match="slide">
<ora:output use="myOutput href="slide{format-number(position(),'00')}.html">
<html>
    <body>
<xsl:apply-templates select="title"/>
       <ul>
```

```
<xsl:apply-templates select="*[not(self::title)]"/>
       </ul>
    </body>
</html>
</ora:output>
</xsl:template>
<xsl:template match="bullet">
    <li><xsl:value-of select="."/></li>
</xsl:template>
<xsl:template match="title">
    <h1><xsl:value-of select="."/></h1>
</xsl:template>
</xsl:stylesheet>
```

Frequently Asked Questions About the XSLT Processor and XSL

This section lists XSL and XSLT Processor questions and answers.

Why Am I Getting an HTML Error in XSL?

I don't know what is wrong here. This is my `news_xsl.xsl` file:

```
<?xml version ="1.0"?>
<xsl:stylesheet  xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
 <HTML>
    <HEAD>
        <TITLE>   Sample Form    </TITLE>
    </HEAD>
       <BODY>
         <FORM>
        <input type="text" name="country" size="15">   </FORM>
       </BODY>
 </HTML>
</xsl:template>
</xsl:stylesheet>


ERROR:End tag 'FORM' does not match the start tag 'input'. Line 14, Position 12
</FORM>-
----------^news.xml
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="news_xsl.xsl"?>
<GREETING/>
```

Answer: Unlike in HTML, in XML every opening or starting tag must have an ending tag. Even the input that you are giving should have a matching ending tag, so you should modify your script like this:

```
<FORM>
<input type="text" name="country" size="15"> </input>
</FORM>
```

Or:

```
<FORM>
<input type="text" name="country" size="15"/>
</FORM>
```

Also, remember that in XML the tags are case sensitive, unlike in HTML.

Is the Output Method "html" Supported in the XSL Parser?

Is the output method `html` supported in the recent version of the XSL parser? I was trying to use the `<BR>` tag with the `<xsl output method="xml"/>` declaration but I got an XSL error message indicating a not well-formed XML document. Then I tried the following output method declaration: `<xsl output method="html"/>` but I got the same result.

Here's a simple XSL stylesheet I was using:

```
<?xml version="1.0"?> <xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <xsl output method="html"/>
<xsl:template match="/">      <HTML>         <HEAD></HEAD>         <BODY>
<P>           Blah blah<BR>           More blah blah<BR>         </P>
</BODY>      </HTML>   </xsl:template>
```

How do I use a not well-formed tag like `<IMG>` or `<BR>` in an XSL stylesheet?

Answer: We fully support all options of `<xsl output>`. The problem here is that your XSL stylesheet must be a well-formed XML document, so everywhere you are using the `<BR>` element, you need to use `<BR/>` instead. The `<xsl output method="html"/>` requests that when the XSLT engine writes out the result of your transformation, it is a proper HTML document. What the XSLT engine reads in must be well-formed XML.

Question: I have a question regarding your reply. I have an XSL stylesheet that preforms XML to HTML conversion. Everything works correctly with the exception of those HTML tags that are not well formed. Using your example if I have something like:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>
......
<input type="text" name="{NAME}" size="{DISPLAY_LENGTH}" maxlength="{LENGTH}">
</input>
......
</xsl:stylesheet>
```

It would render HTML in the format of

```
<HTML>......<input type="text" name="in1" size="10" maxlength="20"/>
......
</HTML>
```

While Internet Explorer can handle this, Netscape cannot. Is there any way to generate completely cross-browser-compliant HTML with XSL?

Answer 2: If you are seeing:

```
<input ... />
```

instead of:

```
<input>
```

then you are likely using the incorrect way of calling `XSLProcessor.processXSL()`, since it appears that it's not doing the HTML output for you. Use:

```
void processXSL(style,sourceDoc,PrintWriter)
```

instead of:

```
DocumentFragment processXSL(style,sourceDoc)
```

and it will work correctly.

Can I Prevent XSL from Returning a Meta-Tag in Netscape 4.0?

I'm using `<xsl output method="html" encoding="iso-8859-1" indent = "no" />`. Is it possible to prevent XSLT from outputting `<META http-equiv="Content-Type" content="text/html; charset=iso-8859-1">` in the `HEAD` element because Netscape 4.0 has difficulties with this statement. It renders the page twice.

Answer: The XSLT 1.0 recommendation says in Section 16.2 ("HTML Output Method") that if there is a HEAD element, then the HTML output method should add a META element immediately after the start-tag of the HEAD element specifying the character encoding actually used.

For example:

<HEAD><META http-equiv="Content-Type" content="text/html; charset=EUC-JP">.
So any XSLT 1.0-compliant engine needs to add this.

How Do I Work Around a Display Bug in the Browser?

Netscape 4.0 has following bug:

When Mozilla hits the meta-encoding tag it stops rendering the page and does a refresh, thereby producing an annoying flickering. I probably have to do a replacement in the servlets Outputstream, but I don't like doing so. Are there any alternatives?

Answer: The only alternatives I can think of are:

- Don't include a <HEAD> section in your HTML page. According to the XSLT specification, this will suppress the inclusion of the <META> tag.
- Don't use method="HTML" for the output. Since it defaults to "HTML", according to the specification for result trees that start with <HTML> (in any mixture of case), you'd have to explicitly set it to method="xml" or method="text".

Neither is pretty, but either one might provide a workaround.

Where Can I Get More Information on XSL Error Messages?

I get the error XSL-1900, exception occurred. What does this mean? How can I find out what caused the exception?

Answer: If you are using Java, you could write exception routines to trap errors. Using tools such as JDeveloper also helps.

The error messages of our components are usually clearer. XSL-1900 indicates possible internal error or incorrect usage.

How Do I Generate the HTML "Less Than" (<) Character?

I am trying to generate an HTML form for inputting data using column names from the user_tab_columns table and the following XSL code:

```
<xsl:template match="ROW">
<xsl:value-of select="COLUMN_NAME"/>
<: lt;INPUT NAME="<xsl:value-of select="COLUMN_NAME"/>>
</xsl:template>
```

although `gt;` is generated as the greater than (>) character, `lt;` is generated as `#60;`. How do I generate the less than (<) character?

Answer: Use the following code:

```
<xsl:text disable-output-escaping="yes">entity-reference</xsl:text>
```

Why Does HTML "<" Conversion Work in oraxsl But Not in XSLSample.java?

I cannot display HTML from XML. In my XML file, I store the HTML snippet in an XML tag:

```
<PRE>
<body.htmlcontent>
<&#60;table width="540" border="0" cellpadding="0"
cellspacing="0">&#60;tr>&#60;td>&#60;font face="Helvetica, Arial"
size="2">&#60;!-- STILL IMAGE GOES HERE -->&#60;img
src="graphics/imagegoeshere.jpg"  width="200" height="175" align="right"
vspace="0" hspace="7">&#60;!-- END STILL IMAGE TAG -->&#60;!-- CITY OR TOWN NAME
GOES FIRST FOLLOWED BY TWO LETTER STATE ABBREVIATION -->&#60;b>City, state
abbreviation&#60;/b> - &#60;!-- CITY OR TOWN NAME ENDS HERE -->&#60;!-- STORY
TEXT STARTS HERE -->Story text goes here.. &#60;!-- STORY TEXT ENDS HERE
-->&#60;/font>&#60;/td>&#60;/tr>&#60;/table>
</body.htmlcontent>
</PRE>
```

I use the following in my XSL:

```
<xsl:value-of select="body.HTMLcontent" disable-output-escaping="yes"/>
```

However, the HTML output

```
<PRE>&#60;</PRE>
```

still appears and all of the HTML tags are displayed in the browser. How do I display the HTML properly?

That doesn't look right. All of the less than (<) characters are `#60;` in the code with an ampersand in front of them. They are still that way when they are displayed in the browser.

Even more confusing is that it works with `oraxsl`, but not with `XSLSample.java`.

Answer: Here's why:

- `oraxsl` internally uses `void XSLProcessor.processXSL (style,source,printwriter);`
- `XSLSample.java` uses `DocumentFragment XSLProcessor.processXSL (style,source);`

The former supports `<xsl:output>` and all options related to writing output that might not be valid XML (including the disable output escaping). The latter is pure XML-to-XML tree returned, so no `<xsl:output>` or disabled escaping can be used since nothing's being output, just a DOM tree fragment of the result is being returned.

Where Can I Find XSLT Examples?

Is there any site which has good examples or short tutorials on XSLT?

Answer: This site is an evolving tutorial on lots of different XML, XSLT, and XPath-related subjects:

`http://zvon.vscht.cz/ZvonHTML/Zvon/zvonTutorials_en.html`

Where Can I Find a List of XSLT Features?

Is there a list of features of the XSLT that the Oracle XDK uses?

Answer: Our version 2 parsers support the W3C Recommendation of w3c XSLT version 1.0, which you can see at `http://www.w3.org/TR/XSLT`.

How Do I Use XSL to Convert an XML Document to Another Form?

I am in the process of trying to convert an XML document from one format to another by means of an XSL (or XSLT) stylesheet. Before incorporating it into my Java code, I tried testing the transformation from the command line:

```
> java oracle.xml.parser.v2.oraxsl jwnemp.xml jwnemp.xsl newjwnemp.xml
```

The problem is that instead of returning the transformed XML file (`newjwnemp.xml`), the above command just returns a file with the XSL code from `jwnemp.xsl` in it. I cannot figure out why this is occurring. I have attached the two input files.

```
<?xml version="1.0"?>
<employee_data>
   <employee_row>
      <employee_number>7950</employee_number>
      <employee_name>CLINTON</employee_name>
      <employee_title>PRESIDENT</employee_title>
      <manager>1111</manager>
      <date_of_hire>20-JAN-93</date_of_hire>
      <salary>125000</salary>
      <commission>1000</commission>
      <department_number>10</department_number>
   </employee_row>
</employee_data>

<?xml version='1.0'?>
<ROWSET xmlns:xsl="HTTP://www.w3.org/1999/XSL/Transform">
   <xsl:for-each select="employee_data/employee_row">
   <ROW>
      <EMPNO><xsl:value-of select="employee_number"/></EMPNO>
      <ENAME><xsl:value-of select="employee_name"/></ENAME>
      <JOB><xsl:value-of select="employee_title"/></JOB>
      <MGR><xsl:value-of select="manager"/></MGR>
      <HIREDATE><xsl:value-of select="date_of_hire"/></HIREDATE>
      <SAL><xsl:value-of select="salary"/></SAL>
      <COMM><xsl:value-of select="commission"/></COMM>
      <DEPTNO><xsl:value-of select="department_number"/></DEPTNO>
   </ROW>
   </xsl:for-each>
</ROWSET>
```

Answer: This is occurring most likely because you have the wrong XSL namespace URI for your `xmlns:xsl="..."` namespace declaration.

If you use the following URI:

`xmlns:xsl="http://www.w3.org/1999/XSL/Transform"`
then everything will work.

If you use `xmlns:xsl="-- any other string here --"` it will do what you're seeing.

Where Can I Find More Information on XSL?

I cannot find anything about using XSL. Can you help? I would like to get an XML and XSL file to show my company what they can expect from this technology. XML alone is not very impressive for users.

Answer: A pretty good starting place for XSL is the following page:

```
http://metalab.unc.edu/xml/books/bible/updates/14.html
```

It provides a simple discussion of the gist of XSL. XSL isn't really anything more than an XML file, so I don't think that it will be anymore impressive to show to a customer. There's also the main Web site for XSL which is:

```
http://www.w3.org/style/XSL/
```

Can the XSL Processor Produce Multiple Outputs?

I recall seeing discussions about the XSL processor producing more than one result from one XML and XSL. How can this can be achieved?

Answer: The XML Parser version 2 release 2.0.2.8 and above supports `<ora:output>` to handle this.

6

XML Schema Processor for Java

This chapter contains the following sections:

Introducing XML Schema

XML Schema was created by the W3C to describe the content and structure of XML documents in XML. It includes the full capabilities of DTDs (Document Type Descriptions) so that existing DTDs can be converted to XML Schema. XML Schemas have additional capabilities compared to DTDs.

How DTDs and XML Schema Differ

Document Type Definition (DTD) is a mechanism provided by XML 1.0 for declaring constraints on XML markup. DTDs allow the specification of the following:

- Which elements can appear in your XML documents
- What elements can be in the elements
- The order the elements can appear

XML Schema language serves a similar purpose to DTDs, but it is more flexible in specifying XML document constraints and potentially more useful for certain applications. See the following section "DTD Limitations".

Consider the XML document:

```
<?XML version="1.0">
<publisher pubid="ab1234">
  <publish-year>2000</publish-year>
     <title>The Cat in the Hat</title>
     <author>Dr. Seuss</author>
     <artist>Ms. Seuss</artist>
     <isbn>123456781111</isbn>
</publisher>
```

Consider a typical DTD for the foregoing XML document:

```
<!ELEMENT publisher (year,title, author+, artist?, isbn)>
<!ELEMENT publish-year (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT isbn (#PCDATA)>
...
```

DTD Limitations

DTDs, also known as XML Markup Declarations, are considered to be deficient in handling certain applications including the following:

- Document authoring and publishing
- Exchange of metadata
- E-commerce
- Inter-database operations

DTD limitations include:

- DTD is not integrated with Namespace technology so users cannot import and reuse code
- DTD does not support data types other than character data, a limitation for describing metadata standards and database schemas
- Applications need to specify document structure constraints more flexibly than the DTD allows for

XML Schema Features

Table 6–1, "XML Schema Features" lists XML Schema features. Note that XML Schema features include DTD features.

Table 6–1 XML Schema Features

XML Schema Feature	DTD
Built-In Data Types	
XML schema specifies a set of builtin datatypes. Some of them are defined and called primitive datatypes, and they form the basis of the type system: string, boolean, float, decimal, double, duration, dateTime, time, date, gYearMonth, gYear, gMonthDat, gMonth, gDay, Base64Binary, HexBinary, anyURI, NOTATION, QName. Others are derived datatypes that are defined in terms of primitive types.	DTDs do not support data types other than character strings.

Table 6–1 XML Schema Features (Cont.)

XML Schema Feature	DTD
User-Defined Data Types	
Users can derive their own datatypes from the builtin data types. There are three ways of datatype derivation: restriction, list and union. Restriction defines a more restricted data type by applying constraining facets to the base type, list simply allows a list of values of its item type, and union defines a new type whose value can be of any of its member types. For example, to specify that the value of publish-year type to be within a specific range: <SimpleType name = "publish-year"> <restriction base="gYear"> <minInclusive value="1970"/> <maxInclusive value="2000"/> </restriction> </SimpleType> The constraining facets are: length, minLength, maxLength, pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive, totalDigits, fractionDigits. Some facets only apply to certain base types. *Note that several facets have been changed since the first release of Oracle XML Schema Processor for Java.*	The publish-year element in the DTD example cannot be constrained further.

Table 6–1 XML Schema Features (Cont.)

XML Schema Feature	DTD
Occurrence Indicators (Content Model or Structure) In XML Schema, the structure (called complexType) of the instance document or an element is defined in terms of model group and attribute group. A model group may further contain model groups or element particles, while attribute group contains attributes. Wildcards can be used in both model group and attribute group to indicate any element or attribute. There are three varies of model group: sequence, all, and choice, representing the sequence, conjunction and disjunction relationships among particles respectively. The range of the number of occurrence of each particle can also be specified. Like the data type, complexType can be derived from other types. The derivation method can be either restriction or extension. The derived type inherits the content of the base type plus corresponding modifications. In addition to inheritance, a type definition can make references to other components. This feature allows a component being defined once and used in many other structures. The type declaration and definition mechanism in XML Schema is much more flexible and powerful than the DTD.	Control by DTDs over the number of child elements in an element are assigned with the following symbols: ■ ? = zero or one. In the foregoing DTD example, artist? implied artist is optional - there may or may not be an artist. ■ * = zero or more ■ + = one or more (in the foregoing DTD example, author+ implies more than one author is possible) ■ (none) = exactly one
Identity Constraints XML Schema extends the concept of XML ID / IDREF mechanism with the declarations of unique, key and keyref. They are part of the type definition and allow not only attributes, but also element contents as keys. Each constraint has a scope within which it holds and the comparison is in terms of their value rather than lexical strings.	
Import/Export Mechanisms (Schema Import, Inclusion and Modification) All components of a schema need not be defined in a single schema file. XML Schema provides a mechanism of assembling multiple schemas. Import is used to integrate schemas of different namespace while inclusion is used to add components of the same namespace. Components can also be modified using redefinition when included.	You cannot use constructs defined in external schemas.

XML Schema can be used to define a class of XML documents. “Instance document” describes an XML document that conforms to a particular schema.

Although these instances and schemas need not exist specifically as “documents”, they are commonly referred to as files. They may exist as any of the following:

- Streams of bytes
- Fields in a database record
- Collections of XML Infoset “Information Items”

See Also:

- http://www.w3.org/TR/xmlschema-0/
- Appendix A, "XDK for Java: Specifications and Quick References"
- *Oracle9i XML API Reference - XDK and Oracle XML DB*

Oracle XML Schema Processor for Java Features

Oracle XML Schema Processor for Java has the following features:

- Supports streaming (SAX) precessing, constant memory usage, and linear processing time.
- Built on the Oracle XML Parser for Java v2
- Fully supports the W3C XML Schema specifications of the Candidate Recommendation (October 24, 2000) and the Recommendation (May 2, 2001).
  - XML Schema Part 0: Primer
  - XML Schema Part 1: Structures
  - XML Schema Part 2: Datatypes

Supported Character Sets

XML Schema Processor for Java supports documents in the following encodings:

- BIG
- EBCDIC-CP-*
- EUC-JP
- EUC-KR
- GB2312

- ISO-2022-JP
- ISO-2022-KR
- ISO-8859-1to -9
- ISO-10646-UCS-2
- ISO-10646-UCS-4
- KOI8-R
- Shift_JIS
- US-ASCII
- UTF-8
- UTF-16

What's Needed to Run XML Schema Processor for Java

To run XML Schema Processor for Java, you need the following:

- Operating Systems: Any OS with Java 1.1.x support
- Java: JDK 1.1.x. or above.

Online Documentation

Documentation for Oracle XML Schema Processor for Java is located in the doc/ directory in your install area.

Release Specific Notes

The readme.html file in the root directory of the archive, contains release specific information including bug fixes, and API additions.

Oracle XML Schema Processor is an early adopter release and is written in Java. It includes the production release of the XML Parser for Java v2.

XML Schema Processor for Java Directory Structure

Table 6–2 lists the directory structure after installing XML Schema Processor for Java.

Table 6–2 Directory Structure for an Installation of XML Schema Processor

Directory and File	Description
license.html	copy of license agreement
readme.html	release and installation notes
doc	directory for documents
lib	directory for class files
sample	directory for sample code files

XML Schema Processor for Java Usage

As shown in Figure 6–1, Oracle's XML Schema processor performs two major tasks:

- A builder assembles schema from schema XML documents
- A validator use the schema to validate instance document.

When building the schema, the builder first calls the DOM Parser to parse the schema XML documents into corresponding DOM trees. It then compiles them into an internal schema object. The validator works as a filter between the SAX parser and your applications for the instance document. The validator takes SAX events of the instance document as input and validates them against the schema. If the validator detects any invalid XML component it sends an error message. The output of the validator is:

- Input SAX events
- Default values it supplies
- Post-Schema Validation (PSV) information

Modes for Schema Validation

The XML Parser supports various modes for schema or DTD validation. The `setValidationMode` method allows different validation parameters to be set. For schema validations, there are these modes available:

- SCHEMA_VALIDATION. With this mode, the schema validator locates and builds schemas and validates the whole or a part of the instance document based on the `schemaLocation` and `noNamespaceSchemaLocation` attributes. See code example `XSDSample.java`.

- SCHEMA_LAX_VALIDATION. The validator tries to validate part or all of the instance document as long as it can find the schema definition. It will not raise an error if it cannot find the definition. See code example `XSDLax.java`.
- SCHEMA_STRICT_VALIDATION. The validator 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.

In addition to the validator to build the schema itself, you can use `XSDBuilder` to build schemas and set it to the validator using `setXMLSchema method`. See code example `XSDSetSchema.java`. By using the `setXMLSchema` method, the validation mode is automatically set to SCHEMA_STRICT_VALIDATION, and both `schemaLocation` and `noNamespaceSchemaLocation` attributes are ignored. You can also change the validation mode to SCHEMA_LAX_VALIDATION.

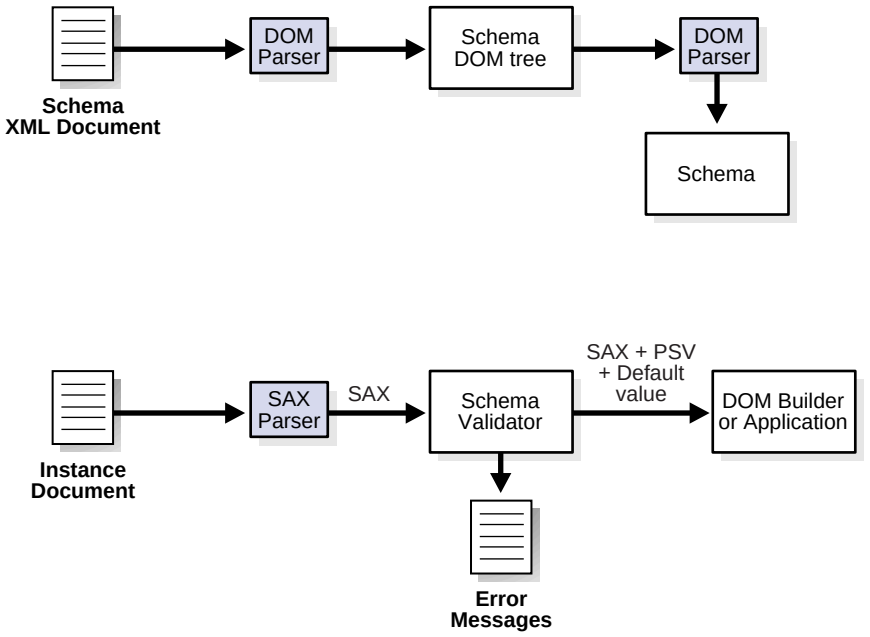
Using the XML Schema API

The API of the XML Schema Processor for Java is simple. You can either use either of the following:

- `setSchemaValidationMode()` in the `DOMParser` as shown in "XML Schema for Java Example 7: XSDSample.java"
- Explicitly build the schema using `XSDBuilder` and set the schema for `XMLParser` as shown in"XML Schema for Java Example 8: XSDSetSchema.java".

There is no clean-up call similar to `xmlclean`. If you need to release all memory and reset the state before validating a new XML document, terminate the context and start over.

Figure 6–1 XML Schema Processor for Java Usage



See Also: *Oracle9i XML API Reference - XDK and Oracle XML DB,* under XDK for Java, XML Schema Processor

How to Run the XML Schema for Java Sample Program

XML Schema Processor for Java directory `sample` contains sample XML applications that illustrate how to use Oracle XML parser with XML Schema Processor for Java. Here are excerpts from the README file:

The sample Java files provided in this directory are:

`XSDSample`, a sample driver that processes XML instance documents.

`XSDSetSchema`, a sample driver to process XML instance documents by overriding the `schemaLocation`.

`XSDLax`, based on `XSDSetSchema`, but uses lax validation mode.

To run the sample program:

1. Execute the program `make` to generate `.class` files.

2. Add `xmlparserv2.jar`, `xschema.jar`, and the current directory to the CLASSPATH.
3. Run the sample program with the `*.xml` files:

```
java XSDSample report.xml
java XSDSetSchema report.xsd report.xml
java XSDLax embeded_xsql.xsd embeded_xsql.xml
```

   XML Schema Processor uses the XML Schema specification from `report.xsd` to validate the contents of `report.xml`.

4. Run the sample program with the `catalogue.xml` file, as follows:

```
java XSDSample   catalogue.xml
java XSDSetSchema cat.xsd  catalogue.xml
```

   XML Schema Processor uses the XML Schema specification from `cat.xsd` to validate the contents of `catalogue.xml`.

5. The following are examples with XML Schema errors:

```
java XSDSample catalogue_e.xml
java XSDSample report_e.xml
```

Makefile for XML Schema Processor for Java

This is the file `Makefile`:

```
# Makefile for sample java files
#
# If not installed in ORACLE_HOME, set ORACLE_HOME to installation root
#
# ======================================================================

.SUFFIXES : .java .class

CLASSES = XSDSample.class XSDSetSchema.class XSDLax.class

# Change it to the appropriate separator based on the OS.
PATHSEP = :

# Assumes that the CLASSPATH contains JDK classes.
MAKE_CLASSPATH =
.$(PATHSEP)$(ORACLE_HOME)/lib/xmlparserv2.jar$(PATHSEP)$(ORACLE_HOME)/lib/xschem
a.jar$(PATHSEP)$(CLASSPATH)
```

```
.java.class:
@javac -classpath "$(MAKE_CLASSPATH)" $<

# make all class files
all: $(CLASSES)

demo: $(CLASSES)
@java -classpath "$(MAKE_CLASSPATH)" XSDSample report.xml > report.out
@java -classpath "$(MAKE_CLASSPATH)" XSDSetSchema report.xsd report.xml >
report.out
@java -classpath "$(MAKE_CLASSPATH)" XSDSample catalogue.xml > catalogue.out
@java -classpath "$(MAKE_CLASSPATH)" XSDSetSchema cat.xsd catalogue.xml >
catalogue.out

@java -classpath "$(MAKE_CLASSPATH)" XSDSample catalogue_e.xml > catalogue_e.out
@java -classpath "$(MAKE_CLASSPATH)" XSDSample report_e.xml > report_e.out

@java -classpath "$(MAKE_CLASSPATH)" XSDLax embeded_xsql.xsd embeded_xsql.xml>
embeded_xsql.out

clean:
@rm -f *.class
@rm -f *.out
```

XML Schema for Java Example 1: cat.xsd

This is the sample XML Schema Definition file that supplies input to the `XSDSetSchema.java` program. XML Schema Processor uses the XML Schema specification from `cat.xsd` to validate the contents of `catalogue.xml`.

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2000/10/XMLSchema"
        targetNamespace="http://www.publishing.org/namespaces/Catalogue"
        elementFormDefault="qualified"
        xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
        xmlns:cat="http://www.publishing.org/namespaces/Catalogue">

    <complexType name="PublicationType">
        <sequence>
            <element name="Title" type="string" minOccurs="1"
maxOccurs="unbounded"/>
            <element name="Author" type="string" minOccurs="1"
maxOccurs="unbounded"/>
            <element name="Date" type="year" minOccurs="1" maxOccurs="1"/>
        </sequence>
```

```
    </complexType>
    <element name="Publication" type="cat:PublicationType" abstract="true"/>
    <element name="Book" substitutionGroup="cat:Publication">
        <complexType>
          <complexContent>
            <extension base="cat:PublicationType">
               <sequence>
                  <element name="ISBN" type="string" minOccurs="1"
maxOccurs="1"/>
                  <element name="Publisher" type="string" minOccurs="1"
maxOccurs="1"/>
               </sequence>
            </extension>
          </complexContent>
        </complexType>
    </element>
    <element name="Magazine" substitutionGroup="cat:Publication">
        <complexType>
          <complexContent>
             <restriction base="cat:PublicationType">
               <sequence>
                  <element name="Title" type="string" minOccurs="1"
maxOccurs="unbounded"/>
                  <element name="Author" type="string" minOccurs="0"
maxOccurs="0"/>
                  <element name="Date" type="year" minOccurs="1" maxOccurs="1"/>
            </sequence>
           </restriction>
         </complexContent>
        </complexType>
    </element>
    <element name="Catalogue">
        <complexType>
            <sequence>
                <element ref="cat:Publication" minOccurs="0"
maxOccurs="unbounded"/>
            </sequence>
        </complexType>
    </element>
</schema>
```

XML Schema for Java Example 2: catalogue.xml

This is the sample XML file that is validated by XML Schema processor against the XML Schema Definition file, cat.xsd, using the program, `XSDSetSchema.java`.

```
<?xml version="1.0"?>
<Catalogue xmlns="http://www.publishing.org/namespaces/Catalogue"
           xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
           xsi:schemaLocation=
                      "http://www.publishing.org/namespaces/Catalogue
                       cat.xsd">
        <Magazine>
                <Title>Natural Health</Title>
                <Date>1999</Date>
        </Magazine>
        <Book>
                <Title>Illusions The Adventures of a Reluctant Messiah</Title>
                <Author>Richard Bach</Author>
                <Date>1977</Date>
                <ISBN>0-440-34319-4</ISBN>
                <Publisher>Dell Publishing Co.</Publisher>
        </Book>
        <Book>
                <Title>The First and Last Freedom</Title>
                <Author>J. Krishnamurti</Author>
                <Date>1954</Date>
                <ISBN>0-06-064831-7</ISBN>
                <Publisher>Harper &amp;amp; Row</Publisher>
        </Book>
</Catalogue>
```

XML Schema for Java Example 3: catalogue_e.xml

When XML Schema Processor processes this sample XML file using `XSDSample.java`, it generates XML Schema errors.

```
<?xml version="1.0"?>
<Catalogue xmlns="http://www.publishing.org/namespaces/Catalogue"
           xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
           xsi:schemaLocation=
                      "http://www.publishing.org/namespaces/Catalogue
                       cat.xsd">
        <Magazine>
                <Title>Natural Health</Title>
                <Date>1999</Date>
        </Magazine>
```

```
        <Book>
                <Title>Illusions The Adventures of a Reluctant Messiah</Title>
                <Author>Richard Bach</Author>
                <Date>July 7, 1977</Date>
                <ISBN>0-440-34319-4</ISBN>
                <Publisher>Dell Publishing Co.</Publisher>
        </Book>
        <Book>
                <Title>The First and Last Freedom</Title>
                <Author>J. Krishnamurti</Author>
                <Date>1954</Date>
                <ISBN>0-06-064831-7</ISBN>
                <ISBN>0-06-064831-7</ISBN>
                <Publisher>Harper &amp;amp; Row</Publisher>
        </Book>
</Catalogue>
```

XML Schema for Java Example 4: report.xml

This is the sample XML file that is validated by XML Schema processor against the XML Schema Definition file, `report.xsd`, using the program, `XSDSetSchema.java`.

```
<purchaseReport
  xmlns="http://www.example.com/Report"
  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"/>
   <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>
```

XML Schema for Java Example 5: report.xsd

This is the sample XML Schema Definition file that inputs XSDSetSchema.java program. XML Schema Processor uses the XML Schema specification from report.xsd to validate the contents of report.xml.

```
<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>
     <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="†{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>
```

XML Schema for Java Example 6: report_e.xml

When XML Schema Processor processes this sample XML file using XSDSample.java, it generates XML Schema errors.

```
<purchaseReport
  xmlns="http://www.example.com/Report"
  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.example.com/Report  report.xsd"
  period="P3M" periodEnding="1999-11-31">

 <regions>
  <zip code="95819">
   <part number="872-AA" quantity="1"/>
   <part number="926-AA" quantity="1"/>
   <part number="833-AA" quantity="1"/>
   <part number="455-BX" quantity="1"/>
  </zip>
  <zip code="63143">
   <part number="455-BX" quantity="4"/>
   <part number="235-JD" quantity="3"/>
  </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>
```

XML Schema for Java Example 7: XSDSample.java

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

import java.net.*;
import java.io.*;
import org.w3c.dom.*;
import java.util.*;

public class XSDSample
```

```
{
   public static void main(String[] args) throws Exception
   {
      if (args.length != 1)
      {
         System.out.println("Usage: java XSDSample <filename>");
         return;
      }
      process (args[0]);
   }

   public static void process (String xmlURI) throws Exception
   {

      DOMParser dp  = new DOMParser();
      URL       url = createURL (xmlURI);

      // Set Schema Validation to true
      dp.setValidationMode(XMLParser.SCHEMA_VALIDATION);
      dp.setPreserveWhitespace (true);

      dp.setErrorStream (System.out);

      try
      {
         System.out.println("Parsing "+xmlURI);
         dp.parse (url);
         System.out.println("The input file <"+xmlURI+"> parsed without
errors");
      }
      catch (XMLParseException pe)
      {
         System.out.println("Parser Exception: " + pe.getMessage());
      }
      catch (Exception e)
      {
         System.out.println("NonParserException: " + e.getMessage());
      }

  }

   // Helper method to create a URL from a file name
   static URL createURL(String fileName)
   {
      URL url = null;
```

```
            try
            {
               url = new URL(fileName);
            }
            catch (MalformedURLException ex)
            {
               File f = new File(fileName);
               try
               {
                  String path = f.getAbsolutePath();
                  // This is a bunch of weird code that is required to
                  // make a valid URL on the Windows platform, due
                  // to inconsistencies in what getAbsolutePath returns.
                  String fs = System.getProperty("file.separator");
                  if (fs.length() == 1)
                  {
                     char sep = fs.charAt(0);
                     if (sep != '/')
                        path = path.replace(sep, '/');
                     if (path.charAt(0) != '/')
                        path = '/' + path;
                  }
                  path = "file://" + path;
                  url = new URL(path);
               }
               catch (MalformedURLException e)
               {
                  System.out.println("Cannot create url for: " + fileName);
                  System.exit(0);
               }
            }
            return url;
         }

      }
```

XML Schema for Java Example 8: XSDSetSchema.java

When this example is run with `cat.xsd` and `catalogue.xml`, XML Schema Processor uses the XML Schema specification from cat.xsd to validate the contents of `catalogue.xml`.

When this example is run with `report.xsd` and `report.xml`, XML Schema Processor uses the XML Schema specification from cat.xsd to validate the contents of `report.xml`.

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

import java.net.*;
import java.io.*;
import org.w3c.dom.*;
import java.util.*;

public class XSDSetSchema
{
   public static void main(String[] args) throws Exception
   {
      if (args.length != 2)
      {
         System.out.println("Usage: java XSDSample <schema_file> <xml_file>");
         return;
      }

      XSDBuilder builder = new XSDBuilder();
      URL    url =  createURL(args[0]);

      // Build XML Schema Object
      XMLSchema schemadoc = (XMLSchema)builder.build(url);
      process(args[1], schemadoc);
   }

   public static void process(String xmlURI, XMLSchema schemadoc)
   throws Exception
   {

      DOMParser dp  = new DOMParser();
      URL       url = createURL (xmlURI);

      // Set Schema Object for Validation
      dp.setXMLSchema(schemadoc);
      dp.setValidationMode(XMLParser.SCHEMA_VALIDATION);
      dp.setPreserveWhitespace (true);

      dp.setErrorStream (System.out);
```

```
      try
      {
         System.out.println("Parsing "+xmlURI);
         dp.parse (url);
         System.out.println("The input file <"+xmlURI+"> parsed without
errors");
      }
      catch (XMLParseException pe)
      {
         System.out.println("Parser Exception: " + pe.getMessage());
      }
      catch (Exception e)
      {
         System.out.println ("NonParserException: " + e.getMessage());
      }

   }

   // Helper method to create a URL from a file name
   static URL createURL(String fileName)
   {
      URL url = null;
      try
      {
         url = new URL(fileName);
      }
      catch (MalformedURLException ex)
      {
         File f = new File(fileName);
         try
         {
            String path = f.getAbsolutePath();
            // This is a bunch of weird code that is required to
            // make a valid URL on the Windows platform, due
            // to inconsistencies in what getAbsolutePath returns.
            String fs = System.getProperty("file.separator");
            if (fs.length() == 1)
            {
               char sep = fs.charAt(0);
               if (sep != '/')
                  path = path.replace(sep, '/');
               if (path.charAt(0) != '/')
                  path = '/' + path;
            }
            path = "file://" + path;
```

```
            url = new URL(path);
         }
         catch (MalformedURLException e)
         {
            System.out.println("Cannot create url for: " + fileName);
            System.exit(0);
         }
      }
      return url;
   }

}
```

XML Schema for Java Example 9: XSDLax.java

Here is a listing of XSDLax.java:

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

import java.net.*;
import java.io.*;
import org.w3c.dom.*;
import java.util.*;

public class XSDLax
{
   public static void main(String[] args) throws Exception
   {
      if (args.length != 2)
      {
         System.out.println("Usage: java XSDSample <schema_file> <xml_file>");
         return;
      }

      XSDBuilder builder = new XSDBuilder();
      URL    url =  createURL(args[0]);

      // Build XML Schema Object
      XMLSchema schemadoc = (XMLSchema)builder.build(url);
      process(args[1], schemadoc);
   }

   public static void process(String xmlURI, XMLSchema schemadoc)
```

```
   throws Exception
   {

      DOMParser dp  = new DOMParser();
      URL       url = createURL (xmlURI);

      // Set Schema Object for Validation
      dp.setXMLSchema(schemadoc);
      dp.setValidationMode(XMLParser.SCHEMA_LAX_VALIDATION);
      dp.setPreserveWhitespace (true);

      dp.setErrorStream (System.out);

      try
      {
         System.out.println("Parsing "+xmlURI);
         dp.parse (url);
         System.out.println("The input file <"+xmlURI+"> parsed without
errors");
      }
      catch (XMLParseException pe)
      {
         System.out.println("Parser Exception: " + pe.getMessage());
      }
      catch (Exception e)
      {
         System.out.println ("NonParserException: " + e.getMessage());
      }

   }

   // Helper method to create a URL from a file name
   static URL createURL(String fileName)
   {
      URL url = null;
      try
      {
         url = new URL(fileName);
      }
      catch (MalformedURLException ex)
      {
         File f = new File(fileName);
         try
         {
            String path = f.getAbsolutePath();
```

```
            // This is a bunch of weird code that is required to
            // make a valid URL on the Windows platform, due
            // to inconsistencies in what getAbsolutePath returns.
            String fs = System.getProperty("file.separator");
            if (fs.length() == 1)
            {
               char sep = fs.charAt(0);
               if (sep != '/')
                  path = path.replace(sep, '/');
               if (path.charAt(0) != '/')
                  path = '/' + path;
            }
            path = "file://" + path;
            url = new URL(path);
         }
         catch (MalformedURLException e)
         {
            System.out.println("Cannot create url for: " + fileName);
            System.exit(0);
         }
      }
      return url;
   }

}
```

XML Schema for Java Example 10: embeded_xsql.xsd

This is the input file for XSDLax.java:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns = "http://xmlns.us.oracle.com/XDK/Example/XSQL/schema"
            targetNamespace =
"http://xmlns.us.oracle.com/XDK/Example/XSQL/schema"
            elementFormDefault="qualified">

<xsd:element name="include-xml">
  <xsd:complexType>
     <xsd:simpleContent>
       <xsd:extension base="xsd:string">
          <xsd:attribute name="href" type="xsd:string"/>
       </xsd:extension>
     </xsd:simpleContent>
  </xsd:complexType>
```

```
</xsd:element>

<xsd:simpleType name="XSQLBool">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="yes"/>
    <xsd:enumeration value="no"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="XSQLTagCase">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="lower"/>
    <xsd:enumeration value="upper"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:element name="query">
  <xsd:complexType>
   <xsd:simpleContent>
    <xsd:extension base="xsd:string">
     <xsd:attribute name="bind-params" type="xsd:string"/>
     <xsd:attribute name="date-format" type="xsd:string"/>
     <xsd:attribute name="error-statement" type="XSQLBool"/>
     <xsd:attribute name="fetch-size" type="xsd:positiveInteger"/>
     <xsd:attribute name="id-attribute" type="xsd:string"/>
     <xsd:attribute name="id-attribute-column" type="xsd:string"/>
     <xsd:attribute name="include-schema" type="XSQLBool"/>
     <xsd:attribute name="max-rows" type="xsd:positiveInteger"/>
     <xsd:attribute name="null-indicator" type="XSQLBool"/>
     <xsd:attribute name="rowset-element" type="xsd:string"/>
     <xsd:attribute name="row-element" type="xsd:string"/>
     <xsd:attribute name="skip-rows" type="xsd:positiveInteger"/>
     <xsd:attribute name="tag-case" type="XSQLTagCase"/>
    </xsd:extension>
   </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>

</xsd:schema>
```

XML Schema for Java Example 11: embeded_xsql.xml

Here is the output file from `XSDLax.java`:

```
<?xml version="1.0" ?>
```

```
<page connection="xdkdemo"
xmlns:xsql="http://xmlns.us.oracle.com/XDK/Example/XSQL/
schema">
  <webpage title=" Search for XDK FAQ">
      <search>
       <xsql:include-xml href="xml/title.xml" />
      </search>
          <content>
            <question>
              <xsql:query fetch-size="50" null-indicator="no">
                     select question from xdkfaq
                     where contains(answer,'{@search}')>0
                  </xsql:query>
              </question>
              <time>
              <xsql:query tag-case="lower" max-rows="20">
                     select to_char(sysdate,'DD-MM-YYY') from dual
                  </xsql:query>
              </time>
          </content>
  </webpage>
</page>
```

7

XML Class Generator for Java

This chapter contains the following sections:

Accessing XML Class Generator for Java

The Oracle XML Class Generator for Java is provided with Oracle9*i*'s XDK for Java. It is located at `$ORACLE_HOME/xdk/java/classgen`. It is also available for download from the OTN site: http://otn.oracle.com/tech/xml.

XML Class Generator for Java: Overview

XML Class Generator for Java creates Java source files from an XML DTD or XML Schema Definition. This is useful in the following situations:

- When an application wants to send an XML message to another application based on agreed-upon DTDs or XML Schemas.
- As the back end of a web form to construct an XML document.

The generated classes can be used to programmatically construct XML documents. XML Class Generator for Java also optionally generates javadoc comments on the generated source files. XML Class Generator for Java requires the XML Parser for Java and the XML Schema Processor for Java. It works in conjunction with XML Parser for Java, which parses the DTD (or XML Schema) and sends the parsed XML document to the Class Generator.

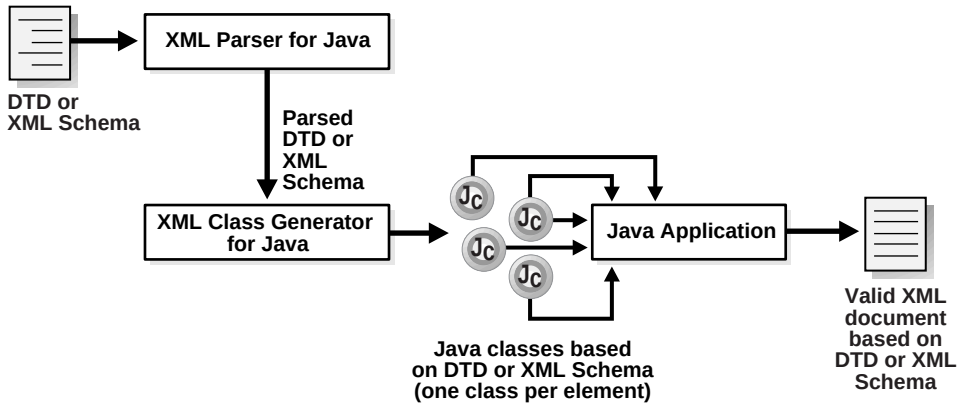
XML Class Generator for Java consists of the following two class generators:

- *DTD Class Generator*
- *XML Schema Class Generator*

These can both be invoked from command line utility, `oracg`.

Figure 7–1 provides an overview of how XML Class Generator for Java is used.

Figure 7–1 XML Class Generator for Java: Overview



Note: The clause, "one class per element" does not apply to the XML Schema Class Generator for Java.

oracg Command Line Utility

The `oracg` command line utility is used to invoke the DTD or Schema Class Generator for Java, depending on the input arguments. Table 7–1 lists the `oracg` arguments.

Table 7–1 Class Generator for Java: oracg Command Line Arguments

oracg Arguments	Description
- help	Print the help message text
- version	Print the release version.
- dtd [-root]	The input file is a DTD file or DTD based XML file.
- schema	The input file is a Schema file or Schema based XML file.
- outputDir	The directory name where Java source is generated.
- package	The package name(s) of the generated java classes.
- comment	Generate comments for the generated java source code.

Class Generator for Java: XML Schema

XML Class Generator for Java's *XML Schema* Class Generator has the following features:

- It generates a Java class for each top level element, that is, global elements `simpleType` element and `complexType` element.
- Classes corresponding to the top level elements, that is, global elements, extend the `CGXSDElement`.
- The type hierarchy among the elements is maintained in the generated Java classes. If the `complexType` or `simpleType` element extends any other `complexType` or `simpleType` element, then the class corresponding to them extends the base type `simpleType` or `complexType` element. Otherwise, they extend the `CGSXDElement` class.

Namespace Features

XML Schema Class Generator also supports the following namespace features:

- *Package Name Creation.* For each namespace, a package is created and corresponds to the elements in the namespace — the Java classes are generated in *that* package.
  - If there is no namespace defined, then the classes are generated in the default package.
  - If `targetNamespace` is specified in the schema, then a package name is required to generate the classes.
- If there is a namespace defined then the user needs to specify the package name through the command line utility. The number of packages specified should match the command line arguments corresponding to the package names.
- *Symbol Spaces.* A single distinct symbol space is used within a given target namespace for each kind of definition and declaration component identified in XML Schema. The exceptions for this is when symbol space is shared between simple type and complex type.

  In a given symbol space, names are unique, but the same name may appear in more than one symbol space without conflict. For example, the same name can appear in both a type definition and an element declaration, without conflict or necessary relation between the two. To resolve this conflict, the classes corresponding to `simpleType` and `complexType` elements are generated in a subdirectory called `types` in the directory corresponding to the package name.

- To avoid conflict, any methods which take the 'type' of an element (corresponding to which there is a generated Java class) as parameter, take the fully resolved name with the package name.

Using XML Class Generator for Java with XML Schema

Figure 7–2 shows the calling sequence used when generating classes with XML Class Generator for Java with XML Schema.

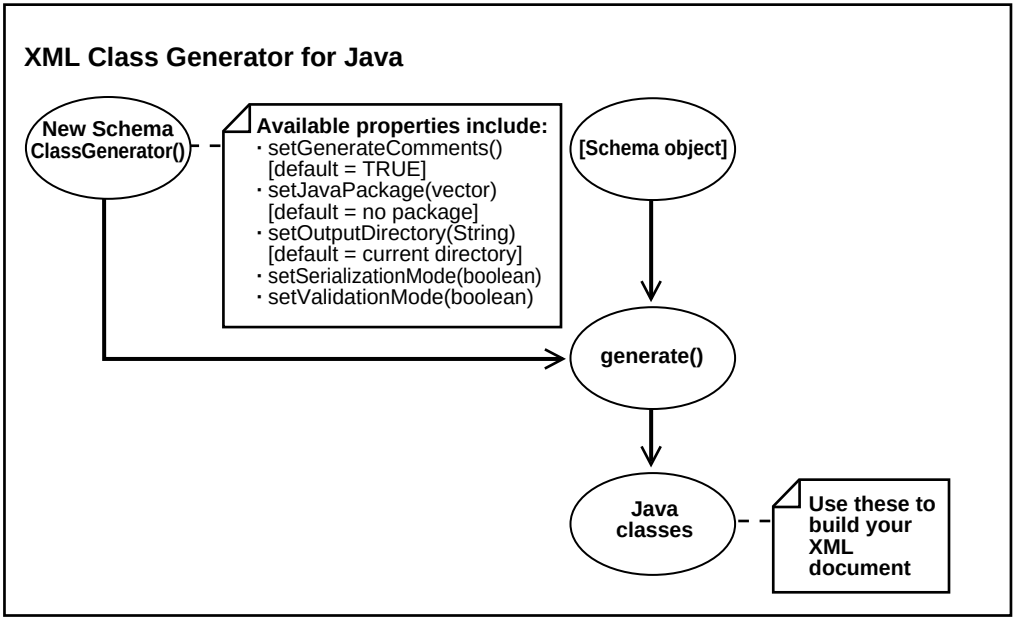
XML Java Class Generator with XML Schema operates as follows:

1. A new `SchemaClassGenerator()` class is initiated and inputs the `generate()` method. The available properties for class, Schema`ClassGenerator()` include:
   - `setGeneraterComments()`, with default = TRUE
   - `setJavaPackage(string)`, with default = no package
   - `setOutputDirectory(string)`, with default = current directory
2. If an XML Schema is used, the Schema object returned using `getDocType()` from the `parseSchema()` method, is also input. See also Figure 4–4, "XML Parser for Java: DOMParser()".
3. The `generate()` method generates Java classes which can then be used to build your XML document.

To generate classes using XML Class Generator for Java with XML Schema, follow the guidelines described in the following sections:

- Generating Top Level Element Classes on page 7-6
- Generating Top Level ComplexType Element Classes on page 7-7
- Generating SimpleType Element Classes on page 7-7

Figure 7–2 Generating Classes Using Class Generator for Java with XML Schema



Generating Top Level Element Classes

The following lists guidelines for using XML Schema Class Generator for Java when generating top level element classes:

- A class corresponding to the element name is generated in the package associated with the namespace.
- The element has a method called `setType` to set the type of the element in the element class. The `setType` takes fully resolved package name to avoid conflict.
- If the element has an inline `simpleType` or `complexType`, a public static class inside the element class is created which follows all the rules specified in the `simpleType`/`complexType`. The name of the public static class, is the element name suffixed by Type. For example, if the element name is `PurchaseOrder` and PurchaseOrder has an inline `complexType` definition, then the public static inner class will have the name `PurchaseOrder_Type`

- The name clash in class names between elements and `complexType` using "Type" as suffix.
- The element name and namespace is stored inside the element class (which could be used for serialization and validation)
- A validate method is provided inside the elements to accept an XML Schema object to validate.
- A print method is provided inside the element to print the node.

Generating Top Level ComplexType Element Classes

The following lists guidelines for using XML Schema Class Generator for Java when generating top level `complexType` element classes:

- If the `complexType` element is a top level element, then a class is generated in the package associated with the namespace. If the `complexType` element extends a base type element, then the class corresponding to the `complexType` element also extends the base Type element. Otherwise, it extends the `CGXSDElement` class.
- The class contains fields corresponding to the attributes. The fields are made protected, so that they can be accessed from subtypes. The fields are added only for the attributes that not present in the base type.
- The class contains methods to set and get attributes.
- For each local element, a public static class is created exactly similar to top level elements, except that it will be completely inside the `complexType` class.

Generating SimpleType Element Classes

The following lists guidelines for using XML Schema Class Generator for Java when generating top level `simpleType` element classes:

- A class is generated for each top level `simpleType` element
- The hierarchy of the `simpleType` element is maintained in the generated class. If the `simpleType` element extends a base class then the class corresponding to the `simpleType` element also extends the base class corresponding to the base element. Otherwise the `simpleType` element extends the `CGXSDElement` class.
- If the `simpleType` element extends the schema data type, then the class extends the class corresponding to the schema data type. For example, if the

base type is a string, then the schema equivalent class is taken as `XSDStringType`, and so on.

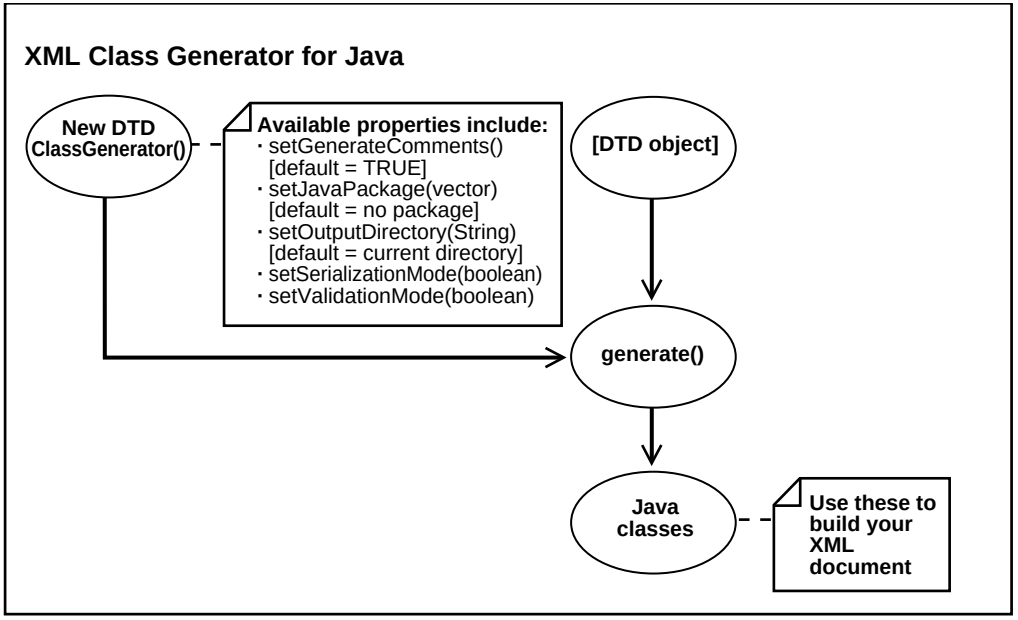
- The class contains a field to store the `simpleType` value.
- The constructor of the `simpleType` element class sets the schema facets.
- The constructor sets the `simpleType` data value (`XSDDataValue`) in the constructor after validating against the facets.

Using XML Class Generator for Java with DTDs

Figure 7–3 shows the calling sequence of XML Java Class Generator with DTDs:

1. A new `DTDClassGenerator()` class is initiated and inputs the `generate()` method. Available properties for class, DTD`ClassGenerator()`are:
   - `setGeneraterComments()`, with default = TRUE
   - `setJavaPackage(string)`, with default = no package
   - `setOutputDirectory(string)`, with default = current directory
2. If a DTD is used, the DTD object returned using `getDocType()` from the `parseDTD()` method, is also input. See also Figure 4–4, "XML Parser for Java: DOMParser()".
3. The `generate()` method generates Java classes which can then be used to build your XML document.

Figure 7–3 Generating Classes Using XML Class Generator for Java and DTDs



See Also:

- Appendix A, "XDK for Java: Specifications and Quick References"
- *Oracle9i XML API Reference - XDK and Oracle XML DB*

Examples Using XML Java Class Generator with DTDs and XML Schema

Table 7–2 lists the example files and directories supplied in $ORACLE_HOME:

Table 7–2 XML Class Generator for Java Example Files

Example File	Description
Makefile	Makefile used to compile and run the demo in Unix.
Make.bat	Makefile used to compile and run the demo in Windows

Table 7–2 XML Class Generator for Java Example Files (Cont.)

Example File	Description
SampleMain.java	Sample application to generate Java source files based on a DTD.
Widl.dtd	Sample DTD.
Widl.xml	Sample XML file based on Widl.dtd.
TestWidl.java	Sample application to construct an XML document using the Java source files generated by SampleMain.
car.xsd	Sample XML Schema
CarDealer.java	Sample application to construct an XML document using the java source generated from car.xsd.
book.xsd	Sample XML Schema
BookCatalogue.java	Sample application to construct an XML document using the Java sources generated from book.xsd
po.xsd	Sample XML Schema
TestPo.java	Sample application to construct an XML document using the Java sources generated from po.xsd.

Running XML Class Generator for Java: DTD Examples

To run the XML Class Generator for Java DTD sample programs, use;

```
make target 'dtd'
```

then follow these steps:

1. Compile and run `SampleMain` to generate the Java source files, using the commands:

   ```
   javac SampleMain.java
   java SampleMain -root WIDL Widl.dtd
   ```

   or

   ```
   java SampleMain Widl.xml
   ```

2. Set the CLASSPATH to contain 'classgen.jar', 'xmlparser.jar', and the current directory.

3. Compile the Java source files generated by SampleMain, that is., BINDING.java, CONDITION.java, REGION.java, SERVICE.java, VARIABLE.java, and WIDL.java, using the command:

   ```
   javac *.java
   ```

4. Run the test application to print the XML Document using the commands:

   ```
   javac TestWidl.java
   java TestWidl
   ```

   The output is stored in Widl_out.txt

Running XML Class Generator for Java: XML Schema Examples

To run the XML Class Generator for Java Schema sample programs, use:

```
make target 'schema'
```

There are three Schema samples: car.xsd, book.xsd, po.xsd

The classes are generated using `oracg` utility. For example, the classes corresponding to car.xsd can be generated from the command line:

```
oracg -c -s car.xsd -p package1
```

The classes are generated in the directory, package1.

When `Makefile` is used to run the schema class generator demo:

- Classes corresponding to car.xsd are generated in directory package1. Demo program, CarDealer.java, tests the generated classes. The output of CarDealer.java is stored in file, car_out.txt.
- Classes corresponding to book.xsd are generated in directory package2. Demo program BookCatalogue.java tests the generated classes. The output is stored in the file, book_out.txt.
- Classes corresponding to po.xsd are generated in directory package3. Demo program TestPo.java tests the generated classes. The output is stored in the file po_out.txt

The following Class Generator using DTD examples are included here:

- XML Class Generator for Java, DTD Example 1a: Application: SampleMain.java
- XML Class Generator for Java, DTD Example 1b: DTD Input — widl.dtd

XML Class Generator for Java, DTD Example 1a: Application: SampleMain.java

```
/**
 * This program generates the classes for a given DTD using
 * XML DTD Class Generator. A DTD file or an XML document which is
 * DTD compliant is given as input parameters to this application.
 */

import java.io.File;
import java.net.URL;
import oracle.xml.parser.v2.DOMParser;
import oracle.xml.parser.v2.DTD;
import oracle.xml.parser.v2.XMLDocument;
import oracle.xml.classgen.DTDClassGenerator;

public class SampleMain
{

   public SampleMain()
   {
   }

   public static void main (String args[])
   {
      // Validate the input arguments
      if (args.length < 1)
      {
         System.out.println("Usage: java SampleMain "+
                            "[-root <rootName>] <fileName>");
         System.out.println("fileName\t   Input file, XML document or " +
                            "external DTD file");
         System.out.println("-root <rootName>   Name of the root Element " +
                            "(required if the input file is an external DTD)");
         return ;
      }

      // ty to open the XML Document or the External DTD File
      try
      {
```

```
// Instantiate the parser
DOMParser   parser = new DOMParser();
XMLDocument doc    = null;
DTD         dtd    = null;

if (args.length == 3)
{
   parser.parseDTD(fileToURL(args[2]), args[1]);
   dtd = (DTD)parser.getDoctype();
}
else
{
   parser.setValidationMode(true);
   parser.parse(fileToURL(args[0]));
   doc = parser.getDocument();
   dtd = (DTD)doc.getDoctype();
}

String doctype_name = null;

if (args.length == 3)
{
   doctype_name = args[1];
}
else
{
   // get the Root Element name from the XMLDocument
   doctype_name = doc.getDocumentElement().getTagName();
}

// generate the Java files...
DTDClassGenerator generator = new DTDClassGenerator();

// set generate comments to true
generator.setGenerateComments(true);

// set output directory
generator.setOutputDirectory(".");

// set validating mode to true
generator.setValidationMode(true);

// generate java src
generator.generate(dtd, doctype_name);
```

```
      }
      catch (Exception e)
      {
         System.out.println ("XML Class Generator: Error " + e.toString());
         e.printStackTrace();
      }
   }

   static public URL fileToURL(String sfile)
   {
      File file = new File(sfile);
      String path = file.getAbsolutePath();
      String fSep = System.getProperty("file.separator");
      if (fSep != null && fSep.length() == 1)
         path = path.replace(fSep.charAt(0), '/');
      if (path.length() > 0 && path.charAt(0) != '/')
         path = '/' + path;
      try
      {
         return new URL("file", null, path);
      }
      catch (java.net.MalformedURLException e)
      {
         // According to the spec this could only happen if the file
         // protocol were not recognized.
         throw new Error("unexpected MalformedURLException");
      }
   }
}
```

XML Class Generator for Java, DTD Example 1b: DTD Input — widl.dtd

The following example, `widl.dtd`, is the DTD file used by SampleMain.java.

```
<!ELEMENT WIDL ( SERVICE | BINDING )* >
<!ATTLIST WIDL
     NAME       CDATA   #IMPLIED
     VERSION (1.0 | 2.0 | ...) "2.0"
     BASEURL    CDATA   #IMPLIED
     OBJMODEL (wmdom | ...) "wmdom"
>

<!ELEMENT SERVICE EMPTY>
<!ATTLIST SERVICE
     NAME       CDATA   #REQUIRED
```

```
     URL        CDATA   #REQUIRED
     METHOD (Get | Post) "Get"
     INPUT      CDATA   #IMPLIED
     OUTPUT     CDATA   #IMPLIED
>

<!ELEMENT BINDING ( VARIABLE | CONDITION | REGION )* >
<!ATTLIST BINDING
     NAME       CDATA   #REQUIRED
     TYPE (Input | Output) "Output"
>

<!ELEMENT VARIABLE EMPTY>
<!ATTLIST VARIABLE
     NAME       CDATA   #REQUIRED
     TYPE (String | String1 | String2) "String"
     USAGE (Function | Header | Internal) "Function"
     VALUE      CDATA   #IMPLIED
     MASK       CDATA   #IMPLIED
     NULLOK    (True | False) #REQUIRED
>

<!ELEMENT CONDITION EMPTY>
<!ATTLIST CONDITION
     TYPE  (Success | Failure | Retry) "Success"
     REF        CDATA   #REQUIRED
     MATCH      CDATA   #REQUIRED
     SERVICE    CDATA   #IMPLIED
>

<!ELEMENT REGION EMPTY>
<!ATTLIST REGION
     NAME       CDATA   #REQUIRED
     START      CDATA   #REQUIRED
     END        CDATA   #REQUIRED
>
```

XML Class Generator for Java, DTD Example 1c: Input — widl.xml

This XML file inputs SampleMain.java and is based on widl.dtd:

```
<?xml version="1.0"?>
<!DOCTYPE WIDL SYSTEM "Widl.dtd">
<WIDL>
    <SERVICE NAME="sname" URL="surl"/>
```

```
    <BINDING NAME="bname"/>
</WIDL>
```

XML Class Generator for Java, DTD Example 1d: TestWidl.java

TestWidl.java constructs an XML document using the Java source files generated by SampleMain.java.

```
/**
 * This is a sample application program which is built using the
 * classes generated by the XML DTD Class Generator. The External DTD
 * File "Widl.dtd" or the XML document which "Widl.xml" which is compliant
 * to  Widl.dtd is used to generate the classes. The application
 * SampleMain.java is used to generate the classes which takes the DTD
 * or XML document as input parameters to generate classes.
 */

import oracle.xml.classgen.CGNode;
import oracle.xml.classgen.CGDocument;
import oracle.xml.classgen.DTDClassGenerator;
import oracle.xml.classgen.InvalidContentException;
import oracle.xml.parser.v2.DTD;

public class TestWidl
{
   public static void main (String args[])
   {
      try
      {
         WIDL w1 = new WIDL();
         DTD dtd = w1.getDTDNode();

         w1.setNAME("WIDL1");
         w1.setVERSION(WIDL.VERSION_1_0);

         SERVICE s1 = new SERVICE("Service1", "Service_URL");
         s1.setINPUT("File");
         s1.setOUTPUT("File");

         BINDING b1 = new BINDING("Binding1");
         b1.setTYPE(BINDING.TYPE_INPUT);

         BINDING b2 = new BINDING("Binding2");
         b2.setTYPE(BINDING.TYPE_OUTPUT);
```

```
      VARIABLE v1 = new VARIABLE("Variable1", VARIABLE.NULLOK_FALSE);
      v1.setTYPE(VARIABLE.TYPE_STRING);
      v1.setUSAGE(VARIABLE.USAGE_INTERNAL);
      v1.setVALUE("value");

      VARIABLE v2 = new VARIABLE("Variable2", VARIABLE.NULLOK_TRUE);
      v2.setTYPE(VARIABLE.TYPE_STRING1);
      v2.setUSAGE(VARIABLE.USAGE_HEADER);

      VARIABLE v3 = new VARIABLE("Variable3", VARIABLE.NULLOK_FALSE);
      v3.setTYPE(VARIABLE.TYPE_STRING2);
      v3.setUSAGE(VARIABLE.USAGE_FUNCTION);
      v3.setMASK("mask");

      CONDITION c1 = new CONDITION("CRef1", "CMatch1");
      c1.setSERVICE("Service1");
      c1.setTYPE(CONDITION.TYPE_SUCCESS);

      CONDITION c2 = new CONDITION("CRef2", "CMatch2");
      c2.setTYPE(CONDITION.TYPE_RETRY);

      CONDITION c3 = new CONDITION("CRef3", "CMatch3");
      c3.setSERVICE("Service3");
      c3.setTYPE(CONDITION.TYPE_FAILURE);

      REGION r1 = new REGION("Region1", "Start", "End");

      b1.addNode(r1);
      b1.addNode(v1);
      b1.addNode(c1);
      b1.addNode(v2);

      b2.addNode(c2);
      b2.addNode(v3);

      w1.addNode(s1);
      w1.addNode(b1);
      w1.addNode(b2);
      w1.validateContent();
      w1.print(System.out);
   }
   catch (Exception e)
   {
      System.out.println(e.toString());
      e.printStackTrace();
```

```
      }
   }
}
```

XML Class Generator for Java, DTD Example 1e: XML Output — widl.out

This XML file, widl.out, is constructed and printed by `TestWidl.java`.

```
<?xml  version = '1.0' encoding = 'ASCII'?>
<!DOCTYPE WIDL SYSTEM
"file:/oracore/java/xml/ORACORE_MAIN_SOLARIS_990115_XMLCLASSGEN/sample/out/WIDL.
dtd">
<WIDL NAME="WIDL1" VERSION="1.0">
   <SERVICE NAME="Service1" URL="Service_URL" INPUT="File" OUTPUT="File"/>
   <BINDING NAME="Binding1" TYPE="Input">
      <REGION NAME="Region1" START="Start" END="End"/>
      <VARIABLE NAME="Variable1" NULLOK="False" TYPE="String" USAGE="Internal"
VALUE="value"/>
      <CONDITION REF="CRef1" MATCH="CMatch1" SERVICE="Service1" TYPE="Success"/>
      <VARIABLE NAME="Variable2" NULLOK="True" TYPE="String1" USAGE="Header"/>
   </BINDING>
   <BINDING NAME="Binding2" TYPE="Output">
      <CONDITION REF="CRef2" MATCH="CMatch2" TYPE="Retry"/>
      <VARIABLE NAME="Variable3" NULLOK="False" TYPE="String2" USAGE="Function"
MASK="mask"/>
   </BINDING>
</WIDL>
```

The following Class Generator using XML Schema examples are included here

- XML Class Generator for Java, Schema Example 1a: XML Schema, car.xsd
- XML Class Generator for Java, Schema Example 1b: Application, CarDealer.java
- XML Class Generator for Java, Schema Example 2a: Schema: book.xsd
- XML Class Generator for Java, Schema Example 2b: BookCatalogue.java
- XML Class Generator for Java, Schema Example 3a: Schema: po.xsd
- XML Class Generator for Java, Schema Example 3b: Application: TestPo.java

XML Class Generator for Java, Schema Example 1a: XML Schema, car.xsd

This sample, car.xsd, is used in an `oracg` command to generate classes. These classes inputs the program, CarDealer.java, which then creates an XML document. The command used is:

```
oracg -c -s car.xsd -p package1
```

See the comments about how this is used, in:

- "XML Class Generator for Java, Schema Example 1b: Application, CarDealer.java" on page 7-20
- "Running XML Class Generator for Java: XML Schema Examples" on page 7-11

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<schema xmlns = "http://www.w3.org/1999/XMLSchema"
targetNamespace = "http://www.CarDealers.com/" elementFormDefault="qualified">
<element name="Car">
  <complexType>
    <element name="Model">
      <simpleType base="string">
        <enumeration value = "Ford"/>
        <enumeration value = "Saab"/>
        <enumeration value = "Audi"/>
      </simpleType>
    </element>
    <element name="Make">
       <simpleType base="string">
         <minLength value = "1"/>
         <maxLength value = "30"/>
       </simpleType>
    </element>
    <element name="Year">
       <complexType content="mixed">
          <attribute name="PreviouslyOwned" type="string" use="required"/>
          <attribute name="YearsOwned" type="integer" use="optional"/>
       </complexType>
    </element>
    <element name="OwnerName" type="string" minOccurs="0"
maxOccurs="unbounded"/>
    <element name="Condition">
       <complexType base="string" derivedBy="extension">
          <attribute name="Automatic">
            <simpleType base="string">
               <enumeration value = "Yes"/>
               <enumeration value = "No"/>
            </simpleType>
          </attribute>
       </complexType>
    </element>
    <element name="Mileage">
```

```
      <simpleType base="integer">
        <minInclusive value="0"/>
        <maxInclusive value="20000"/>
      </simpleType>
    </element>
    <attribute name="RequestDate" type="date"/>
  </complexType>
 </element>
</schema>
```

XML Class Generator for Java, Schema Example 1b: Application, CarDealer.java

```
/**
 * This is a sample application program that creates an XMl document. It is
 * built using the classes generated by XML Schema Class Generator. XML
 * Schema "car.xsd", is used to generate the classes using the oracg
 * command line utility. The classes are generated in a package called
 * package1 which is specified as command line option. The following
 * oracg command line options are used to generate the classes:
 * oracg -c -s car.xsd -p package1
 */

import oracle.xml.classgen.CGXSDElement;
import oracle.xml.classgen.SchemaClassGenerator;
import oracle.xml.classgen.InvalidContentException;
import oracle.xml.parser.v2.XMLOutputStream;
import java.io.OutputStream;

import package1.*;

public class CarDealer
{
   static OutputStream output  = System.out;
   static XMLOutputStream out = new XMLOutputStream(output);

   public static void main(String args[])
   {
      CarDealer cardealer = new CarDealer();
      try
      {
         Car.Car_Type ctype = new Car.Car_Type();
         ctype.setRequestDate("02-09-00");
         Car.Car_Type.Model model = new Car.Car_Type.Model();
         Car.Car_Type.Model.Model_Type modelType =
                  new Car.Car_Type.Model.Model_Type("Ford");
```

```
model.setType(modelType);
ctype.addModel(model);

Car.Car_Type.Make make = new Car.Car_Type.Make();
Car.Car_Type.Make.Make_Type makeType =
         new Car.Car_Type.Make.Make_Type("F150");
make.setType(makeType);
ctype.addMake(make);

Car.Car_Type.Year year = new Car.Car_Type.Year();
Car.Car_Type.Year.Year_Type yearType =
        new Car.Car_Type.Year.Year_Type();
yearType.addText("1999");

year.setType(yearType);
ctype.addYear(year);

Car.Car_Type.OwnerName owner1 = new Car.Car_Type.OwnerName();
owner1.setType("Joe Smith");
ctype.addOwnerName(owner1);

Car.Car_Type.OwnerName owner2 = new Car.Car_Type.OwnerName();
owner2.setType("Bob Smith");
ctype.addOwnerName(owner2);

String str = "Small dent on the car's right bumper.";
Car.Car_Type.Condition condition = new Car.Car_Type.Condition();
Car.Car_Type.Condition.Condition_Type conditionType =
        new Car.Car_Type.Condition.Condition_Type(str);

Car.Car_Type.Condition.Condition_Type.Automatic automatic =
      new Car.Car_Type.Condition.Condition_Type.Automatic("Yes");
conditionType.setAutomatic(automatic);

condition.setType(conditionType);
ctype.addCondition(condition);

Car.Car_Type.Mileage mileage = new Car.Car_Type.Mileage();
Car.Car_Type.Mileage.Mileage_Type mileageType =
        new Car.Car_Type.Mileage.Mileage_Type("10000");
mileage.setType(mileageType);
ctype.addMileage(mileage);

Car car = new Car();
car.setType(ctype);
```

```
            car.print(out);

            out.writeNewLine();
            out.flush();
         }
         catch(InvalidContentException e)
         {
            System.out.println(e.getMessage());
            e.printStackTrace();
         }
         catch(Exception e)
         {
            System.out.println(e.getMessage());
            e.printStackTrace();
         }
      }
}
```

XML Class Generator for Java, Schema Example 2a: Schema: book.xsd

This sample schema, book.xsd, is used in an `oracg` command to generate classes. The classes then input the program, CarDealer.java, which creates an XML document. The `oracg` command is:

```
oracg -c -s book.xsd -p package2
```

See the comments about how this is used, in:

- "XML Class Generator for Java, Schema Example 2b: BookCatalogue.java" on page 7-23
- "Running XML Class Generator for Java: XML Schema Examples" on page 7-11

```
<?xml version="1.0"?>
<schema xmlns = "http://www.w3.org/1999/XMLSchema"
        targetNamespace = "http://www.somewhere.org/BookCatalogue"
        xmlns:cat = "http://www.somewhere.org/BookCatalogue"
        elementFormDefault="qualified">

    <complexType name="Pub">
      <sequence>
        <element name="Title" type="cat:titleType" maxOccurs="*"/>
        <element name="Author" type="string" maxOccurs="*"/>
        <element name="Date" type="date"/>
      </sequence>
```

```
      <attribute name="language" type="string" use="default" value="English"/>
    </complexType>

    <complexType name="titleType" base="string" derivedBy="extension">
      <attribute name="old" type="string" use="default" value="false"/>
    </complexType>

    <element name="Catalogue" type="cat:Pub"/>
</schema>
```

XML Class Generator for Java, Schema Example 2b: BookCatalogue.java

```
/**
 * This is a sample application program built using the
 * classes generated by XML Schema Class Generator. XML
 * Schema "book.xsd" is used to generate the classes using the oracg
 * command line utility. The classes are generated in a package called
 * package2 which is specified as command line option. The following
 * oracg command line options are used to generate the classes:
 * oracg -c -s book.xsd -p package2
 */

import oracle.xml.classgen.SchemaClassGenerator;
import oracle.xml.classgen.CGXSDElement;
import oracle.xml.classgen.InvalidContentException;
import oracle.xml.parser.v2.XMLOutputStream;
import java.io.OutputStream;

import package2.*;

public class BookCatalogue
{
   static OutputStream output  = System.out;
   static XMLOutputStream out = new XMLOutputStream(output);

   public static void main(String args[])
   {
      BookCatalogue bookCatalogue = new BookCatalogue();
      try
      {
         Pub pubType = new Pub();

         TitleType titleType = new TitleType("Natural Health");
         titleType.setOld("true");
```

```
            Pub.Title title = new Pub.Title();
            title.setType(titleType);
            pubType.addTitle(title);

            Pub.Author author = new Pub.Author();
            author.setType("Richard> Bach");
            pubType.addAuthor(author);

            Pub.Date date = new Pub.Date();
            date.setType("1977");
            pubType.addDate(date);
            pubType.setLanguage("English");

            Catalogue catalogue = new Catalogue();
            catalogue.setType(pubType);

            catalogue.print(out);
            out.writeNewLine();
            out.flush();
         }
         catch(InvalidContentException e)
         {
            System.out.println(e.getMessage());
            e.printStackTrace();
         }
         catch(Exception e)
         {
            System.out.println(e.getMessage());
            e.printStackTrace();
         }
      }
   }
```

XML Class Generator for Java, Schema Example 3a: Schema: po.xsd

This sample schema, po.xsd, is used in an `oracg` command to generate classes. The classes then input the program, TestPo.java, which creates an XML document. The `oracg` command used is:

```
oracg -c -s po.xsd -p package3
```

See the comments about how this is used, in:

- "XML Class Generator for Java, Schema Example 3b: Application: TestPo.java" on page 7-26
- "Running XML Class Generator for Java: XML Schema Examples" on page 7-11


```
<?xml version="1.0" encoding="ISO-8859-1"?>
<schema xmlns = "http://www.w3.org/1999/XMLSchema"
        targetNamespace = "http://www.somewhere.org/PurchaseOrder"
        xmlns:po = "http://www.somewhere.org/PurchaseOrder">

<element name="comment" type="string"/>

<element name="PurchaseOrder">
  <complexType>
      <element name="shipTo" type="po:Address"/>
      <element name="billTo" type="po:Address"/>
      <element ref="po:comment" minOccurs="0"/>
      <element name="items" type="po:Items"/>
      <attribute name="orderDate" type="date"/>
      <attribute name="shipDate" type="date"/>
      <attribute name="receiveDate" type="date"/>
  </complexType>
</element>

<complexType name="Address">
  <element name="name" type="string"/>
  <element name="street" type="string"/>
  <element name="city" type="string"/>
  <element name="zip" type="decimal"/>
  <attribute name="country" type="NMTOKEN"
      use="fixed" value="US"/>
</complexType>

<complexType name="Items">
   <element name="item" minOccurs="0" maxOccurs="unbounded">
      <complexType>
        <element name="productName" type="string"/>
        <element name="quantity" type="int"/>
        <element name="price" type="decimal"/>
        <element name="shipDate" type="date" minOccurs='0'/>
        <attribute name="partNum" type="string"/>
      </complexType>
   </element>
</complexType>
```

```
</schema>
```

XML Class Generator for Java, Schema Example 3b: Application: TestPo.java

```
/**
 * This is a sample application program which is built using the
 * classes generated by XML Schema Class Generator. XML
 * Schema "po.xsd" is used to generate the classes using the oracg
 * command line utility. The classes are generated in a package called
 * package3 which is specified as command line option. The following
 * oracg command line options are used to generate the classes:
 * oracg -c -s po.xsd -p package3
 */

import oracle.xml.classgen.CGXSDElement;
import oracle.xml.classgen.SchemaClassGenerator;
import oracle.xml.classgen.InvalidContentException;
import oracle.xml.parser.v2.XMLOutputStream;
import java.io.OutputStream;
import package3.*;

public class TestPo
{
   static OutputStream output = System.out;
   static XMLOutputStream out = new XMLOutputStream(output);

   public static void main (String args[])
   {
      TestPo testpo = new TestPo();
      try
      {
         // Create Purchase Order
         PurchaseOrder po = new PurchaseOrder();

         // Create Purchase Order Type
         PurchaseOrder.PurchaseOrder_Type poType =
                new PurchaseOrder.PurchaseOrder_Type();

         // Set purchase order date
         poType.setOrderDate("December 17, 2000");
         poType.setShipDate("December 19, 2000");
         poType.setReceiveDate("December 21, 2000");

         // Create a PurchaseOrder shipTo item
```

```
PurchaseOrder.PurchaseOrder_Type.ShipTo shipTo =
       new PurchaseOrder.PurchaseOrder_Type.ShipTo();

// Create Address
Address address = new Address();

// Create the Name for the address and add
// it to addresss
Address.Name name = new Address.Name();
name.setType("Mary Smith");
address.addName(name);

// Create the Stree name for the address and add
// it to the address
Address.Street street = new Address.Street();
street.setType("Laurie Meadows");
address.addStreet(street);

// Create the city name for the address and add
// it to the address
Address.City city = new Address.City();
city.setType("San Mateo");
address.addCity(city);

// Create the zip name for the address and add
// it to the address
Address.Zip zip = new Address.Zip();
zip.setType(new Double("11208"));
address.addZip(zip);

// Set the address of the shipTo object
shipTo.setType(address);
// Add the shipTo to the Purchase Type object
poType.addShipTo(shipTo);

// Create a Purchase Order BillTo item
PurchaseOrder.PurchaseOrder_Type.BillTo billTo =
      new PurchaseOrder.PurchaseOrder_Type.BillTo();

// Create a billing Address
Address billingAddress = new Address();

// Create the name for billing address, set the
// name and add it to the billing address
Address.Name name1 = new Address.Name();
```

```
name1.setType("John Smith");
billingAddress.addName(name1);

// Create the street name for the billing address,
// set the street name value and add it to the
// billing address
Address.Street street1 = new Address.Street();
street1.setType("No 1. North Broadway");
billingAddress.addStreet(street1);

// Create the City name for the address, set the
// city name value and add it to the billing address
Address.City city1 = new Address.City();
city1.setType("New York");
billingAddress.addCity(city1);

// Create the Zip for the address, set the zip
// value and add it to the billing address.
Address.Zip zip1 = new Address.Zip();
zip1.setType(new Double("10006"));
billingAddress.addZip(zip1);

// Set the type of the billTo object to billingAddress
billTo.setType(billingAddress);

// Add the billing address to the PurchaseOrder type
poType.addBillTo(billTo);

PurchaseOrder.PurchaseOrder_Type.Items pItem =
      new PurchaseOrder.PurchaseOrder_Type.Items();

Items items = new Items();
Items.Item item = new Items.Item();
Items.Item.Item_Type itemType = new Items.Item.Item_Type();

Items.Item.Item_Type.ProductName pname =
    new Items.Item.Item_Type.ProductName();
pname.setType("Perfume");
itemType.addProductName(pname);

Items.Item.Item_Type.Quantity qty =
    new Items.Item.Item_Type.Quantity();
qty.setType(new Integer("1"));
itemType.addQuantity(qty);
```

```
            Items.Item.Item_Type.Price price =
                new Items.Item.Item_Type.Price();
            price.setType(new Double("69.99"));
            itemType.addPrice(price);

            Items.Item.Item_Type.ShipDate sdate =
                new Items.Item.Item_Type.ShipDate();
            sdate.setType("Feb 14. 2000");
            itemType.addShipDate(sdate);

            itemType.setPartNum("ITMZ411");

            item.setType(itemType);
            items.addItem(item);

            pItem.setType(items);

            poType.addItems(pItem);

            // Set the type of the Purchase Order object to
            // Purchase Order Type
            po.setType(poType);
            po.print(out);

            out.writeNewLine();
            out.flush();
         }
         catch (InvalidContentException e)
         {
            System.out.println(e.getMessage());
            e.printStackTrace();
         }
         catch (Exception e)
         {
            System.out.println(e.toString());
            e.printStackTrace();
         }
      }
   }
```

Frequently Asked Questions About the Class Generator for Java

This section lists XML Java Class Generator questions and answers.

How Do I Install the XML Class Generator for Java?

Answer: The Class Generator is packaged as part of the XDK and so you do not have to download it separately. The `CLASSPATH` should be set to include `classgen.jar`, `xmlparserv2.jar`, and `xschema.jar` which are located in the `lib/` directory and not in the `bin/` directory.

What Does the XML Class Generator for Java Do?

What does the XML Class Generator for Java do? How do I use it to get XML data?

Answer: The XML Class Generator for Java creates Java source files from an XML DTD. This is useful when you need an application to send an XML message to another application based on an agreed-upon DTD or as the back end of a Web form to construct an XML document. Using these classes, Java applications can construct, validate, and print XML documents that comply with the input DTD. The Class Generator works in conjunction with the Oracle XML Parser for Java version 2, which parses the DTD and passes the parsed document to the class generator.

To get XML data, first, get the data from the database using JDBC ResultSets. Then, instantiate objects using the classes generated by the XML Class Generator.

Which DTDs Are Supported?

Does XML Java Class Generator support any kind of DTD?

Answer: Yes, it supports any kind of DTD that is XML 1.0 compliant.

Why Do I Get a "Classes Not Found" Error?

Why do I get a "Class Not Found" error when running XML Class Generator samples?

Answer: Correct your `CLASSPATH` to include `classgen.jar`, `xmlparserv2.jar`, and `xschema.jar`.

In XML Class Generator, How Do I Create the Root Object More Than Once?

I generated, from a DTD, a set of Java classes with the Class Generator. Since then, I've tried to create a Java application that uses these classes to create an XML file from data passed as arguments. I cannot create the root object, the object derived from CGDocument, more than one time because I obtain the following error message:

```
oracle.xml.parser.XMLDOMException: Node doesn't belong to the current document
```

How do I handle the star operator (*)? When the application starts I do not know how many times the element will appear. Thus I do not build a static loop where I make a sequence of `element.addNode()`. The problem is that some of these will be empty and I will obtain an XML document with a set of empty elements with empty attributes.

Answer: You can create subsequent XML docs by calling the constructor each time. A well-formed XML document is not permitted to have more than one root node, therefore you cannot use the star operator (*) on the element you are designating as the document root.

How Can I Create XML Files from Scratch Using the DOM API?

I want to create an XML file using the DOM API. I do not want to create the XML file by typing in the text editor:

```
<xml>
  <future>is great</future>
</xml>
```

Instead, I want to create it using the DOM API. There are several examples of manipulating an XML file using the DOM once there is an input file, but not the other way round. That is, of creating the XML file from scratch (when there is no input file) using the DOM, once you know the tagnames and their values.

Answer: The simplest way is download XML Class Generator for Java and give it a DTD of the XML document you want. It will create the DOM classes to programmatically create XML documents. There are samples included with the software.

Can I Create an XML Document in a Java Class?

I need to create an XML document in a Java class as follows

```
<?xml version = '1.0' encoding = 'WINDOWS-1252'?>
    <root>
     <listing>
       <one> test </one>
       <two> test </two>
     </listing>
    </root>
```

Can I use the `XMLDocument` class to create an XML document? I know about the XML SQL Utility, but that only creates XML based on SQL queries, which is not what I am after on this occasion. Do you have an example of how to do this?

Answer: The XML Class Generator, available as part of the Oracle XDK for Java, does the job nicely. The XDKs are also available with Oracle9*i* and Oracle9*i* Application Server products. The Class Generator generates Java classes for each element in your DTD. These classes can then be used to dynamically construct an XML document at runtime. The Class Generator download contains sample code.

8

XML SQL Utility (XSU)

This chapter contains the following sections:

What Is XML SQL Utility (XSU)?

XML has become the format for data interchange. At the same time, a substantial amount of business data resides in object-relational databases. It is therefore necessary to have the ability to transform this relational data to XML.

XML SQL Utility (XSU) enables you to do this as follows:

- XSU can transform data retrieved from object-relational database tables or views into XML.
- XSU can extract data from an XML document, and using a canonical mapping, insert the data into appropriate columns or attributes of a table or a view.
- XSU can extract data from an XML document and apply this data to updating or deleting values of the appropriate columns or attributes.

Generating XML from the Database

For example, on the XML generation side, when given the query `SELECT * FROM emp`, XSU queries the database and returns the results as the following XML document:

```
<?xml version='1.0'?>
<ROWSET>
  <ROW num="1">
    <EMPNO>7369</EMPNO>
    <ENAME>Smith</ENAME>
    <JOB>CLERK</JOB>
    <MGR>7902</MGR>
    <HIREDATE>12/17/1980 0:0:0</HIREDATE>
    <SAL>800</SAL>
    <DEPTNO>20</DEPTNO>
  </ROW>
  <!-- additional rows ... -->
</ROWSET>
```

Storing XML in the Database

Going the other way, given the XML document preceding, XSU can extract the data from it and insert it into the `scott.emp` table in the database.

Accessing XSU Functionality

XML SQL Utility functionality can be accessed in the following ways:

- Through a Java API
- Through a PL/SQL API
- Through a Java command line front end

See Also:

- *Oracle9i XML API Reference - XDK and Oracle XML DB*

XSU Features

XSU can perform the following tasks:

- Generate XML documents from any SQL query. XSU virtually supports all the datatypes supported in the Oracle9*i* database server.
- Dynamically generate DTDs (Document Type Definitions).
- During generation, perform simple transformations, such as modifying default tag names for the `ROW` element. You can also register an XSL transformation which is then applied to the generated XML documents as needed.
- Generate XML documents in their string or DOM representations.
- Insert XML into database tables or views. XSU can also update or delete records records from a database object, given an XML document.
- Easily generate complex nested XML documents. XSU can also store them in relational tables by creating object views over the flat tables and querying over these views. Object views can create structured data from existing relational data using Oracle*8i* and Oracle9*i*'s object-relational infrastructure.

XSU Oracle9*i* New Features

Starting in Oracle9*i*, XSU can also perform the following tasks:

- Generates XML Schema given an SQL Query.
- Generates XML as a stream of SAX2 callbacks.
- Supports XML attributes during generation. This provides an easy way to specify that a particular column or group of columns should be mapped to an XML attribute instead of an XML element.
- SQL identifier to XML identifier escaping. Sometimes column names are not valid XML tag names. To avoid this you can either alias all the column names or turn on tag escaping.

Note: Oracle9i introduced the `DBMS_XMLGen` PL/SQL supplied package. This package provides the functionality previously available with `DBMS_XMLQuery`. `DBMS_XMLGen` is built into the database code, hence, it provides better performance.

XSU Supports XMLType

From Oracle9i Release 2 (9.2), XSU supports XMLType. Using XSU with XMLType is useful if, for example, you have XMLType columns in objects or tables.

See Also: *Oracle9i XML Database Developer's Guide - Oracle XML DB*, in particular, the chapter on Generating XML, for examples on using XSU with XMLType.

XSU Dependencies and Installation

Dependencies

XML SQL Utility (XSU) needs the following components:

- *Database connectivity -- JDBC drivers*. XSU can work with any JDBC driver but is optimized for Oracle JDBC drivers. Oracle does not make any guarantee or provide support for the XSU running against non-Oracle databases.
- *XML Parser -- Oracle XML Parser, Version2*. Oracle XML Parser, version 2 is included in Oracle8*i* and Oracle9*i*, and is also available as part of the XSU install (XDK for Java) downloadable from the Oracle Technology Network (OTN) Web site.

Installation

XML SQL Utility (XSU) is packaged with Oracle8*i* (8.1.7 and later) and Oracle9*i*. XSU is made up of three files:

- `$ORACLE_HOME/rdbms/jlib/xsu12.jar` -- Contains all the Java classes which make up XSU. `xsu12` requires `JDK1.2.x` and `JDBC2.x`. This is the XSU version loaded into Oracle9*i*.
- `$ORACLE_HOME/rdbms/jlib/xsu111.jar` -- Contains the same classes as `xsu12.jar`, except that `xsu111` requires `JDK1.1.x` and `JDBC1.x`.

- `$ORACLE_HOME/rdbms/admin/dbmsxsu.sql` -- This is the SQL script that builds the XSU PL/SQL API. `xsu12.jar` needs to be loaded into the database before `dbmsxsu.sql` is executed.

By default the Oracle9*i* installer installs XSU on the hard drive in the locations specified earlier. It also loads it into the database.

If during initial installation you choose to not install XSU, you can install it later, but the installation becomes less simple. To install XSU later, first install XSU and its dependent components on your system. You can accomplish this using Oracle Installer. Next perform the following steps:

1. If you have not yet loaded XML Parser for Java in the database, go to `$ORACLE_HOME/xdk/lib`. Here you will find `xmlparserv2.jar` that you need to load into the database. To do this, see "Loading JAVA Classes" in the *Oracle9i Java Stored Procedures Developer's Guide*
2. Go to `$ORACLE_HOME/admin` and run `catxsu.sql`

> **Note:** XML SQL Utility (XSU) is part of the XDK for Java and is also available on OTN at: `http://otn.oracle.com/tech/xml`

XML SQL Utility and the Bigger Picture

XML SQL Utility (XSU) is written in Java, and can live in any tier that supports Java.

XML SQL Utility in the Database

The Java classes which make up XSU can be loaded into Java-enabled Oracle8*i* or later. Also, XSU contains a PL/SQL wrapper that publishes the XSU Java API to PL/SQL, creating a PL/SQL API. This way you can:

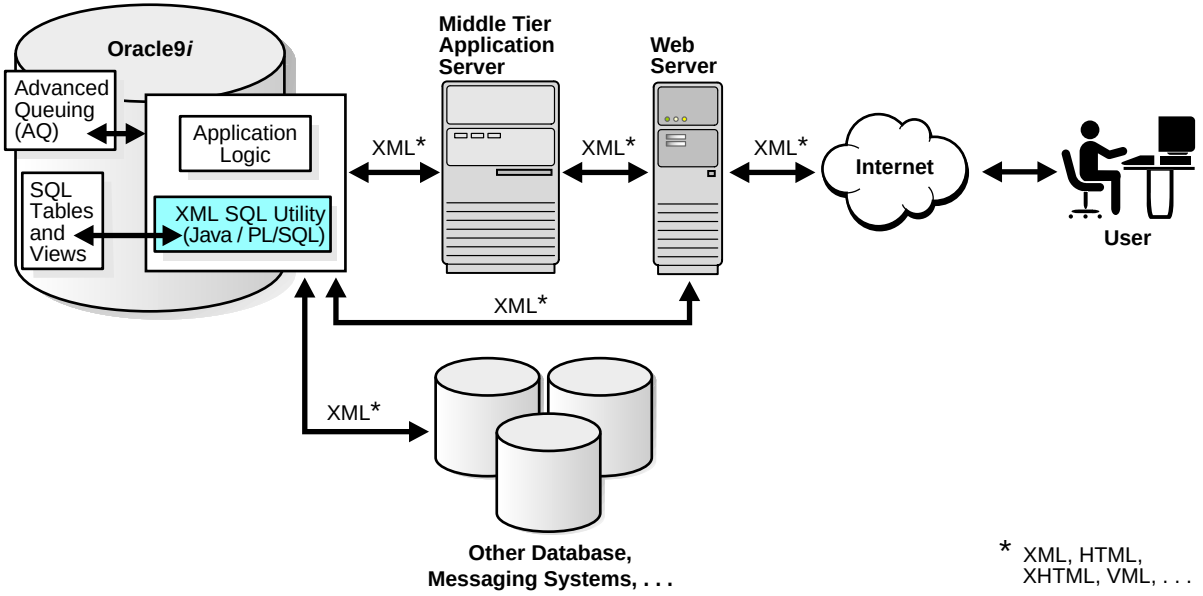
- Write new Java applications that run inside the database and that can directly access the XSU Java API
- Write PL/SQL applications that access XSU through its PL/SQL API
- Access XSU functionality directly through SQL

> **Note:** To load and run Java code inside the database you need a Java-enabled Oracle8*i* or later server.

Figure 8–1 shows the typical architecture for such a system. XML generated from XSU running in the database, can be placed in advanced queues in the database to be queued to other systems or clients. The XML can be used from within stored procedures in the database or shipped outside through web servers or application servers.

Note: In Figure 8–1, all lines are bi-directional. Since XSU can *generate* as well as *save* data, data can come from various sources to XSU running inside the database, and can be put back in the appropriate database tables.

Figure 8–1 Running XML SQL Utility in the Database



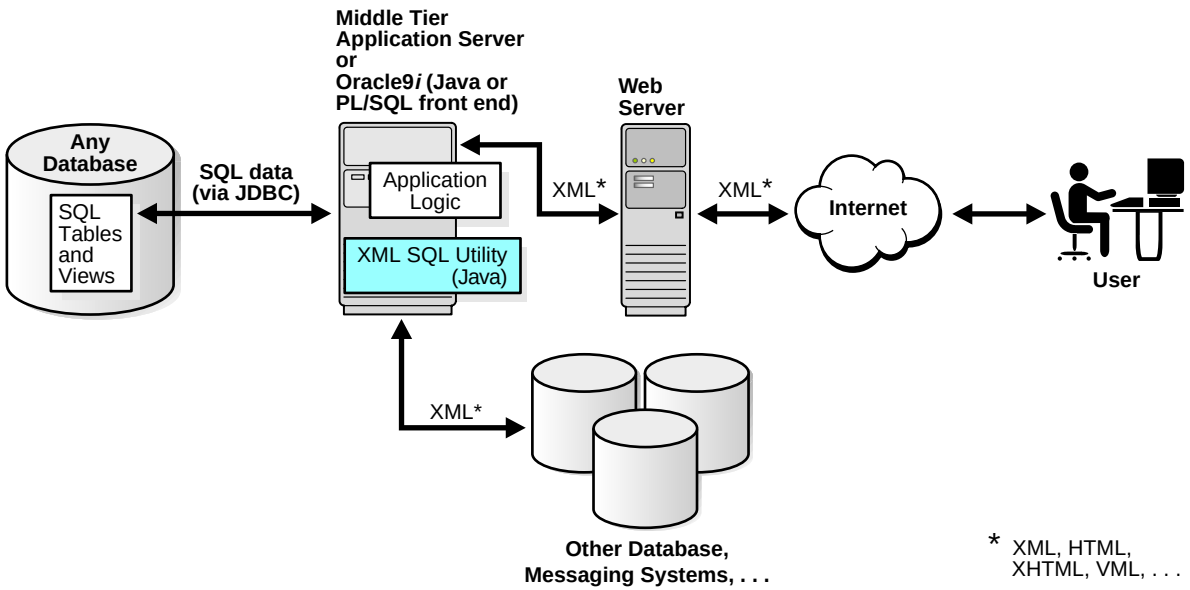
XML SQL Utility in the Middle Tier

Your application architecture may need to use an application server in the middle tier, separate from the database. The application tier can be an Oracle database, Oracle9*i* Application Server, or a third party application server that supports Java programs.

You may want to generate XML in the middle tier, from SQL queries or ResultSets, for various reasons. For example, to integrate different JDBC data sources in the middle tier. In this case you could install the XSU in your middle tier and your Java programs could make use of XSU through its Java API.

Figure 8–2, shows how a typical architecture for running XSU in a middle tier. In the middle tier, data from JDBC sources is converted by XSU into XML and then sent to Web servers or other systems. Again, the whole process is bi-directional and the data can be put back into the JDBC sources (database tables or views) using XSU. If an Oracle database itself is used as the application server, then you can also use the PL/SQL front-end instead of Java.

Figure 8–2 Running XML SQL Utility in the MIddle Tier



XML SQL Utility in a Web Server

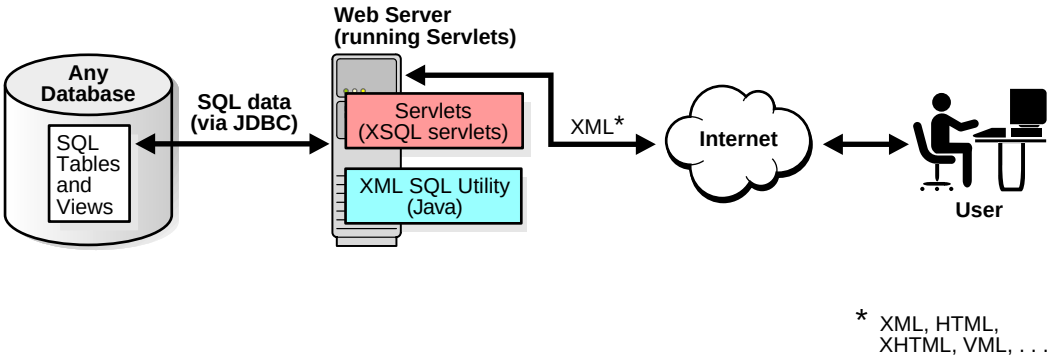
XSU can live in the Web server, as long as the Web server supports Java servlets. This way you can write Java servlets that use XSU to accomplish their task.

XSQL servlet does just this. XSQL servlet is a standard servlet provided by Oracle. It is built on top of XSU and provides a template-like interface to XSU functionality.

If XML processing in the Web server is your goal, you should probably use the XSQL servlet, as it will spare you from the intricate servlet programming.

> **See:** Chapter 9, "XSQL Pages Publishing Framework" for information about using XSQL Servlet.

Figure 8–3 Running XML SQL Utility in a Web Server



XML SQL Utility in the Client Tier

XML SQL Utility can be also installed on a client system, where you can write Java programs that use XSU. You can also use XSU directly through its command line front end.

SQL-to-XML and XML-to-SQL Mapping Primer

As described earlier, XML SQL Utility transforms data retrieved from object-relational database tables or views into XML. XSU can also extract data from an XML document, and using a specified mapping, insert the data into appropriate columns or attributes of a table or a view in the database. This section describes the canonical mapping or transformation used to go from SQL to XML or vice versa.

Default SQL-to-XML Mapping

Consider table `emp`:

```
CREATE TABLE emp
(
   EMPNO NUMBER,
```

```
    ENAME VARCHAR2(20),
    JOB VARCHAR2(20),
    MGR  NUMBER,
    HIREDATE DATE,
    SAL NUMBER,
    DEPTNO NUMBER
);
```

XSU can generate the following XML document by specifying the query, `select * from emp`:

```
<?xml version='1.0'?>
<ROWSET>
  <ROW num="1">
    <EMPNO>7369</EMPNO>
    <ENAME>Smith</ENAME>
    <JOB>CLERK</JOB>
    <MGR>7902</MGR>
    <HIREDATE>12/17/1980 0:0:0</HIREDATE>
    <SAL>800</SAL>
    <DEPTNO>20</DEPTNO>
  </ROW>
  <!-- additional rows ... -->
</ROWSET>
```

In the generated XML, the rows returned by the SQL query are enclosed in a `ROWSET` tag to constitute the `<ROWSET>` element. This element is also the root element of the generated XML document.

- The `<ROWSET>` element contains one or more `<ROW>` elements.
- Each of the `<ROW>` elements contain the data from one of the returned database table rows. Specifically, each `<ROW>` element contains one or more elements whose names and content are those of the database columns specified in the `SELECT` list of the SQL query.
- These elements, corresponding to database columns, contain the data from the columns.

SQL-to-XML Mapping Against Object-Relational Schema

Next we describe this mapping but against an object-relational schema. Consider the following type, `AddressType`. Its an object type whose attributes are all scalar types and is created as follows:

```
CREATE TYPE AddressType AS OBJECT (
```

```
   STREET VARCHAR2(20),
   CITY   VARCHAR2(20),
   STATE  CHAR(2),
   ZIP    VARCHAR2(10)
);
/
```

The following type, `EmployeeType`, is also an object type but it has an `EMPADDR` attribute that is of an object type itself, specifically, `AddressType`. `Employee Type` is created as follows:

```
CREATE TYPE EmployeeType AS OBJECT
(
  EMPNO NUMBER,
  ENAME VARCHAR2(20),
  SALARY NUMBER,
  EMPADDR AddressType
);
/
```

The following type, `EmployeeListType`, is a collection type whose elements are of the object type, `EmployeeType`. `EmployeeListType` is created as follows:

```
CREATE TYPE EmployeeListType AS TABLE OF EmployeeType;
/
```

Finally, `dept` is a table with, among other things, an object type column and a collection type column -- `AddressType` and `EmployeeListType` respectively.

```
CREATE TABLE dept
(
  DEPTNO NUMBER,
  DEPTNAME VARCHAR2(20),
  DEPTADDR AddressType,
  EMPLIST  EmployeeListType
)
NESTED TABLE EMPLIST STORE AS EMPLIST_TABLE;
```

Assume that valid values are stored in table, `dept`. For the query `select * from dept`, XSU generates the following XML document:

```
<?xml version='1.0'?>
<ROWSET>
   <ROW num="1">
    <DEPTNO>100</DEPTNO>
    <DEPTNAME>Sports</DEPTNAME>
```

```
    <DEPTADDR>
      <STREET>100 Redwood Shores Pkwy</STREET>
      <CITY>Redwood Shores</CITY>
      <STATE>CA</STATE>
      <ZIP>94065</ZIP>
    </DEPTADDR>
    <EMPLIST>
      <EMPLIST_ITEM num="1">
         <EMPNO>7369</EMPNO>
         <ENAME>John</ENAME>
         <SALARY>10000</SALARY>
         <EMPADDR>
           <STREET>300 Embarcadero</STREET>
           <CITY>Palo Alto</CITY>
           <STATE>CA</STATE>
           <ZIP>94056</ZIP>
         </EMPADDR>
      </EMPLIST_ITEM>
       <!-- additional employee types within the employee list -->
    </EMPLIST>
  </ROW>
  <!-- additional rows ... -->
</ROWSET>
```

As in the last example, the mapping is canonical, that is, <ROWSET> contains <ROW>s that contain elements corresponding to the columns. As before, the elements corresponding to scalar type columns simply contain the data from the column.

Mapping Complex Type Columns to XML

Things get more complex with elements corresponding to a complex type column. For example, <DEPTADDR> corresponds to the DEPTADDR column which is of object type ADDRESS. Consequently, <DEPTADDR> contains subelements corresponding to the attributes specified in the type ADDRESS. These subelements can contain data or sub-elements of their own, again depending if the attribute they correspond to is of a simple or complex type.

Mapping Collections to XML

When dealing with elements corresponding to database collections, things are yet different. Specifically, the <EMPLIST> element corresponds to the EMPLIST column which is of a EmployeeListType collection type. Consequently, the <EMPLIST>

element contains a list of <EMPLIST_ITEM> elements each corresponding to one of the elements of the collection.

Other observations to make about the preceding mapping are:

- 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 left out altogether.
- If a top level scalar column name starts with the at sign (@) character, then the particular column is mapped to an XML attribute instead of an XML element.

Customizing the Generated XML: Mapping SQL to XML

Often, one needs to generate XML with a specific structure. Since the desired structure may differ from the default structure of the generated XML document, it is desirable to have some flexibility in this process. You can customize the structure of a generated XML document using one of the following methods:

- "Source Customizations"
- "Mapping Customizations"
- "Post-Generation Customizations"

Source Customizations

Source customizations are done by altering the query or database schema. The simplest and most powerful source customizations include the following:

- *In the database schema*, create an object-relational view that maps to the desired XML document structure.
- *In your query:*
  - Use cursor subqueries, or cast-multiset constructs to get nesting in the XML document which comes from a flat schema.
  - Alias column / attribute names to get the desired XML element names.
  - Alias top level scalar type columns with identifiers which begin with the at sign (@) to have them map to an XML attribute instead of an XML element. For example, `select empno as "@empno",... from emp`, results in an XML document where the <ROW> element has an attribute EMPNO.

Mapping Customizations

XML SQL Utility enables you to modify the mapping it uses to transform SQL data into XML. You can make any of the following SQL to XML mapping changes:

- Change or omit the <ROWSET> tag.
- Change or omit the <ROW> tag.
- Change or omit the attribute num. This is the cardinality attribute of the <ROW> element.
- Specify the case for the generated XML element names.
- Specify that XML elements corresponding to elements of a collection, should have a cardinality attribute.
- Specify the format for dates in the XML document.
- Specify that null values in the XML document should be indicated using a nullness attribute, rather then by omission of the element.

Post-Generation Customizations

Finally, if the desired customizations cannot be achieved with the foregoing methods, you can write an XSL transformation and register it with XSU. While there is an XSLT registered with the XSU, XSU can apply the XSLT to any XML it generates.

Default XML-to-SQL Mapping

XML to SQL mapping is just the reverse of the SQL to XML mapping.

See Also: "Default SQL-to-XML Mapping" on page 8-8.

Consider the following differences when mapping from XML to SQL, compared to mapping from SQL to XML:

- When going from XML to SQL, the XML attributes are ignored. Thus, there is really no mapping of XML attributes to SQL.
- When going from SQL to XML, mapping is performed from the resultset created by the SQL query to XML. This way the query can span multiple database tables or views. What gets formed is a single resultset which is then converted into XML. This is not the case when going from XML to SQL, where:

- To insert one XML document into multiple tables or views, you must create an object-relational view over the target schema.
- If the view is not updatable, one work around is to use `INSTEAD-OF-INSERT` triggers.

If the XML document does not perfectly map into the target database schema, there are three things you can do:

- **Modify the Target.** Create an object-relational view over the target schema, and make the view the new target.
- **Modify the XML Document.** Use XSLT to transform the XML document. The XSLT can be registered with XSU so that the incoming XML is automatically transformed, before any mapping attempts are made.
- **Modify XSU's XML-to-SQL Mapping.** You can instruct XSU to perform case insensitive matching of the XML elements to database columns or attributes.
  - If not the default (ROW), you can tell XSU to use the name of the element corresponding to a database row.
  - You can instruct XSU on which date format to use when parsing dates in the XML document.

How XML SQL Utility Works

This section describes how XSU works when performing the following tasks:

Selecting with XSU

XSU generation is simple. SQL queries are executed and the resultset is retrieved from the database. Metadata about the resultset is acquired and analyzed. Using the mapping described in "Default SQL-to-XML Mapping" on page 8-8, the SQL result set is processed and converted into an XML document.

Inserting with XSU

To insert the contents of an XML document into a particular table or view, XSU first retrieves the metadata about the target table or view. Based on the metadata, XSU generates a SQL `INSERT` statement. XSU extracts the data out of the XML document and binds it to the appropriate columns or attributes. Finally the statement is executed.

For example, assume that the target table is `dept` and the XML document is the one generated from `dept`.

See Also: "Default SQL-to-XML Mapping" on page 8-8.

XSU generates the following `INSERT` statement.

```
INSERT INTO Dept (DEPTNO, DEPTNAME, DEPTADDR, EMPLIST) VALUES (?,?,?,?)
```

Next, the XSU parses the XML document, and for each record, it binds the appropriate values to the appropriate columns or attributes, and executes the statement:

```
DEPTNO <- 100
DEPTNAME <- SPORTS
DEPTADDR <- AddressType('100 Redwood Shores Pkwy','Redwood Shores',
                        'CA','94065')

EMPLIST <- EmployeeListType(EmployeeType(7369,'John',100000,
            AddressType('300 Embarcadero','Palo Alto','CA','94056'),...)
```

Insert processing can be optimized to insert in batches, and commit in batches. More detail on batching can be found in the section on "Insert Processing Using XSU (Java API)" on page 8-38.

Updating with XSU

Updates and deletes differ from inserts in that they can affect more than one row in the database table. For inserts, each `ROW` element of the XML document can affect at most, one row in the table, if there are no triggers or constraints on the table.

However, with both updates and deletes, the XML element could match more than one row if the matching columns are not key columns in the table. For updates, you must provide a list of key columns which XSU needs to identify the row to update. For example, to update the `DEPTNAME` to `SportsDept` instead of `Sports`, you can have an XML document such as:

```
<ROWSET>
  <ROW num="1">
    <DEPTNO>100</DEPTNO>
    <DEPTNAME>SportsDept</DEPTNAME>
  </ROW>
</ROWSET>
```

and supply the DEPTNO as the key column. This would result in the following UPDATE statement:

```
UPDATE DEPT SET DEPTNAME = ? WHERE DEPTNO = ?
```

and bind the values,

```
DEPTNO <- 100
DEPTNAME <- SportsDept
```

For updates, you can also choose to update only a set of columns and not all the elements present in the XML document. See also, "Update Processing Using XSU (Java API)" on page 8-40.

Deleting with XSU

For deletes, you can choose to give a set of key columns for the delete to identify the rows. If the set of key columns are not given, then the DELETE statement tries to match all the columns given in the document. For an XML document:

```
<ROWSET>
  <ROW num="1">
    <DEPTNO>100</DEPTNO>
    <DEPTNAME>Sports</DEPTNAME>
    <DEPTADDR>
      <STREET>100 Redwood Shores Pkwy</STREET>
      <CITY>Redwood Shores</CITY>
      <STATE>CA</STATE>
      <ZIP>94065</ZIP>
    </DEPTADDR>
  </ROW>
  <!-- additional rows ... -->
</ROWSET>
```

To delete, XSU fires off a DELETE statement (one for each ROW element) which looks like the following:

```
DELETE FROM Dept WHERE DEPTNO = ? AND DEPTNAME = ? AND DEPTADDR = ?
binding,
```

```
DEPTNO <- 100
DEPTNAME <- Sports
DEPTADDR <-  AddressType('100 Redwood Shores Pkwy','Redwood
City','CA','94065')
```

See also, "Delete Processing Using XSU (Java API)" on page 8-43.

Using the XSU Command Line Front End, OracleXML

XSU comes with a simple command line front end which gives you quick access to XML generation and insertion.

> **Note:** In Oracle9i, the XSU front end does not publish the update and delete functionality.

The XSU command line options are provided through the Java class, `OracleXML`. Invoke it by calling:

```
java OracleXML
```

This prints the front end usage information. To run the XSU command line front end, first specify where the executable is located. Add the following to your `CLASSPATH`:

- XSU Java library (`xsu12.jar` or `xsu111.jar`)

Also, since XSU depends on Oracle XML Parser and JDBC drivers, make the location of these components known. To do this, the `CLASSPATH` must include the locations of:

- Oracle XML Parser Java library (`xmlparserv2.jar`)
- JDBC library (`classes12.jar` if using `xsu12.jar` or `classes111.jar` if using `xsu111.jar`)
- A JAR file for XMLType.

Generating XML Using the XSU Command Line

For XSU generation capabilities, use the XSU `getXML` parameter. For example, to generate an XML document by querying the `emp` table in the `scott` schema, use:

```
java OracleXML getXML -user "scott/tiger" "select * from emp"
```

This performs the following tasks:

- Connects to the current default database
- Executes the query `select * from emp`
- Converts the result to XML
- Displays the result

The `getXML` parameter supports a wide range of options which are explained in the following section.

XSU's OracleXML getXML Options

Table 8–1 lists the OracleXML `getXML` options:

Table 8–1 XSU's OracleXML getXML Options

getXML Option	Description
-user "*username*/*password*"	Specifies the user name and password to connect to the database. If this is not specified, the user defaults to `scott/tiger`. Note that he connect string is also being specified, the user name and password can be specified as part of the connect string.
-conn "*JDBC_connect_string*"	Specifies the JDBC database connect string. By default the connect string is: "`jdbc:oracle:oci8:@`"):
-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 `rowset` tag (the tag that encloses all the XML elements corresponding to the records returned by the query). The default `rowset` tag is `ROWSET`. Specifying an empty string for the `rowset` tells the XSU to completely omit the `rowset` element.
-rowTag "*tag_name*"	Specifies the `row` tag (the tag used to enclose the data corresponding to a database row). The default row tag is `ROW`. Specifying an empty string for the row tag tells the XSU to completely omit the row tag.
-rowIdAttr "*row_id-attribute-name*"	Names the attribute of the `ROW` element keeping track of the cardinality of the `rows`. By default this attribute is called `num`. Specifying an empty string (that is, "") as the `rowID` attribute will tell the XSU to omit the attribute.
-rowIdColumn "*row Id column name*"	Specifies that the value of one of the scalar columns from the query should be used as the value of the `rowID` attribute.
-collectionIdAttr "*collection id attribute name*"	Names the attribute of an XML list element keeping track of the cardinality of the elements of the list (Note: the generated XML lists correspond to either a cursor query, or collection). Specifying an empty string (that is, "") as the `rowID` attribute will tell the XSU to omit the attribute.
-useNullAttrId	Tells the XSU to use the attribute `NULL (TRUE/FALSE)` to indicate the nullness of an element.

Table 8–1 XSU's OracleXML getXML Options (Cont.)

getXML Option	Description
-styleSheet "*stylesheet URI*"	Specifies the stylesheet in the XML PI (Processing Instruction).
-stylesheetType "*stylesheet type*"	Specifies the stylesheet type in the XML PI (Processing Instruction).
-errorTag "*error tag name*"	Secifies the error tag -- the tag to enclose error messages which are formatted into XML.
-raiseNoRowsException	Tells the XSU to raise an exception if no rows are returned.
-maxRows "*maximum number of rows*"	Specifies the maximum number of rows to be retrieved and converted to XML.
-skipRows "*number of rows to skip*"	Specifies the number of rows to be skipped.
-encoding "*encoding name*"	Specifies the character set encoding of the generated XML.
-dateFormat "*date format*"	Specifies the date format for the date values in the XML document.
-fileName "*SQL query fileName*" | sql query	Specifies the file name which contains the query or specify the query itself.
-useTypeForCollElemTag	Use type name for coll-elem tag (by default XSU uses the *column-name_item*.
-setXSLTRef *"URI"*	Set the XSLT external entity reference.
-useLowerCase | useUpperCase	Generate lowercase or uppercase tag names, respectively. The default is to match the case of the SQL object names from which generating the tags.
-withEscaping	There are character which are legal in SQL object names but illegal in XML tags. This option means that if such a character is encountered, it is escaped rather then an exception being thrown.
-raiseException	By default the XSU catches any error and produces the error XML doc. This changes this behavior so the XSU actually throws the raised Java exception.

Inserting XML Using XSU's Command Line (putXML)

To insert an XML document into the `emp` table in the `scott` schema, use the following syntax:

```
java OracleXML putXML -user "scott/tiger" -fileName "/tmp/temp.xml" "emp"
```

This performs the following tasks:

- Connects to the current database
- Reads the XML document from the given file
- Parses it, matches the tags with column names
- Inserts the values appropriately in to the `emp` table

Note: The XSU command line front end, `putXML`, currently only publishes XSU `insert` functionality. It may be expanded in future to also publish XSU `update and delete` functionality.

XSU OracleXML putXML Options

Table 8–2 lists the `putXML` options:

Table 8–2 XSU's OracleXML putXML Options

putXML Options	Description
-user "*username*/*password*"	Specifies the user name and password to connect to the database. If this is not specified, the user defaults to `scott/tiger`. Note that he connect string is also being specified, the user name and password can be specified as part of the connect string.
-conn "*JDBC_connect_string*"	Specifies the JDBC database connect string. By default the connect string is: "`jdbc:oracle:oci8:@`"):
-batchSize "*batching_size*"	Specifies the batch size, which control the number of rows which are batched together and inserted in a single trip to the database. Batching improves performance.
-commitBatch "*commit_size*"	Specifies the number of inserted records after which a commit is to be executed. Note that if the autocommit is `true` (default), then setting the `commitBatch` has no consequence.
-rowTag "*tag_name*"	Specifies the `row` tag (the tag used to enclose the data corresponding to a database row). The default row tag is `ROW`. Specifying an empty string for the `row` tag tells the XSU that no row enclosing tag is used in the XML document.
-dateFormat "*date_format*"	Specifies the date format for the date values in the XML document.
-ignoreCase	Makes the matching of the column names with tag names case insensitive (for example, "EmpNo" will match with "EMPNO" if ignoreCase is on).
-fileName "*file_name*" | -URL "*URL*" | -xmlDoc "*xml_document*"	Specifies the XML document to insert. The fileName option specifies a local file, the URL specifies a URL to fetch the document from and the xmlDoc option specifies the XML document as a string on the command line.
-tableName *"table"*	The name of the table to put the values into.
-withEscaping	If SQL to XML name escaping was used when generating the doc, then this will turn on the reverse mapping.
-setXSLT *"URI"*	XSLT to apply to XML doc before inserting.
-setXSLTRef *"URI"*	Set the XSLT external entity reference.

XSU Java API

The following two classes make up the XML SQL Utility Java API:

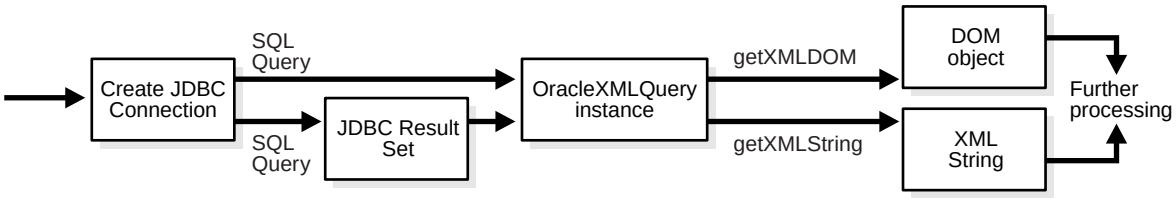
- XSU API for XML generation: `oracle.xml.sql.query.OracleXMLQuery`
- XSU API for XML `save`, `insert`, `update`, and `delete`: `oracle.xml.sql.dml.OracleXMLSave`

Generating XML with XSU's OracleXMLQuery

The `OracleXMLQuery` class makes up the XML generation part of the XSU Java API. Figure 8–4 illustrates the basic steps you need to take when using OracleXMLQuery to generate XML:

1. Create a connection.
2. Create an `OracleXMLQuery` instance by supplying an SQL string or a `ResultSet` object.
3. Obtain the result as a DOM tree or XML string.

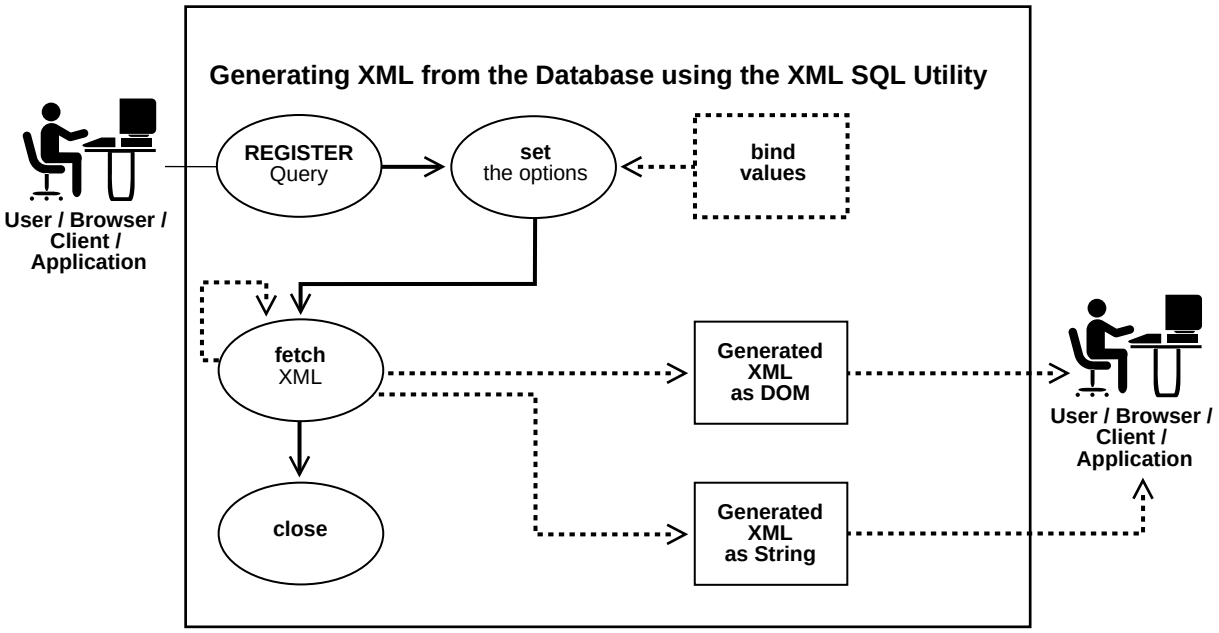
Figure 8–4 Generating XML With XML SQL Utility for Java: Basic Steps



Generating XML from SQL Queries Using XSU

The following examples illustrate how XSU can generate an XML document in its DOM or string representation given a SQL query. See Figure 8–5.

Figure 8–5 Generating XML With XML SQL Utility



XSU Generating XML Example 1: Generating a String from Table emp (Java)

1. Create a connection

Before generating the XML you must create a connection to the database. The connection can be obtained by supplying the JDBC connect string. First register the Oracle JDBC class and then create the connection, as follows

```
// import the Oracle driver..
import oracle.jdbc.driver.*;

// Load the Oracle JDBC driver
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

// Create the connection.
Connection conn =
 DriverManager.getConnection("jdbc:oracle:oci8:@","scott","tiger");
```

Here, the connection is done using OCI8's JDBC driver. You can connect to the `scott` schema supplying the password `tiger`. It connects to the current

database (identified by the ORA_SID environment variable). You can also use the JDBC thin driver to connect to the database. The thin driver is written in pure Java and can be called from within applets or any other Java program.

See Also: *Oracle9i Java Developer's Guide* for more details.

- *Connecting With the Thin Driver.* Here is an example of connecting using the JDBC thin driver:

```
// Create the connection.
Connection conn =
DriverManager.getConnection("jdbc:oracle:thin:@dlsun489:1521:ORCL",
                            "scott","tiger");
```

  The thin driver requires you to specify the host name (dlsun489), port number (*1521*), and the Oracle SID (*ORCL)*, which identifies a specific Oracle instance on the machine.

- *No Connection Needed When Run In the Server.* When writing server side Java code, that is, when writing code that will run on the server, you need not establish a connection using a username and password, since the server-side internal driver runs within a default session. You are already connected. In this case call the `defaultConnection()` on the `oracle.jdbc.driver.OracleDriver()` class to get the current connection, as follows:

```
import oracle.jdbc.driver.*;

// Load the Oracle JDBC driver
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
Connection conn =  new oracle.jdbc.driver.OracleDriver
().defaultConnection ();
```

The remaining discussion either assumes you are using an OCI8 connection from the client or that you already have a connection object created. Use the appropriate connection creation based on your needs.

2. Creating an OracleXMLQuery Class Instance

Once you have registered your connection, create an `OracleXMLQuery` class instance by supplying a SQL query to execute as follows:

```
// import the query class in to your class
import oracle.xml.sql.query.OracleXMLQuery;
```

```
OracleXMLQuery qry = new OracleXMLQuery (conn, "select * from emp");
```

You are now ready to use the query class.

3. **Obtain the result as a DOM tree or XML string**

   - *DOM Object Output*. If, instead of a string, you wanted a DOM object, you can simply request a DOM output as follows:

     ```
     org.w3c.DOM.Document domDoc = qry.getXMLDOM();
     ```

     and use the DOM traversals.

   - *XML String Output*. You can get an XML string for the result by:

     ```
     String xmlString = qry.getXMLString();
     ```

Here is a complete listing of the program to extract (generate) the XML string. This program gets the string and prints it out to standard output:

```
import oracle.jdbc.driver.*;
import oracle.xml.sql.query.OracleXMLQuery;
import java.lang.*;
import java.sql.*;

// class to test the String generation!
class testXMLSQL {

   public static void main(String[] argv)
   {

     try{
      // create the connection
      Connection conn  = getConnection("scott","tiger");

      // Create the query class.
      OracleXMLQuery qry = new OracleXMLQuery(conn, "select * from emp");

      // Get the XML string
      String str = qry.getXMLString();

      // Print the XML output
      System.out.println(" The XML output is:\n"+str);
      // Always close the query to get rid of any resources..
     qry.close();
     }catch(SQLException e){
      System.out.println(e.toString());
```

```
    }
  }

  // Get the connection given the user name and password..!
  private static Connection getConnection(String username, String password)
    throws SQLException
  {
     // register the JDBC driver..
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());


     // Create the connection using the OCI8 driver
      Connection conn =
       DriverManager.getConnection("jdbc:oracle:oci8:@",username,password);

     return conn;
  }
}
```

How to Run This Program

To run this program, carry out the following:

1. Store this in a file called `testXMLSQL.java`
2. Compile it using `javac`, the Java compiler
3. Execute it by specifying: `java testXMLSQL`

You must have the `CLASSPATH` pointing to this directory for the Java executable to find the class. Alternatively use various visual Java tools including Oracle JDeveloper to compile and run this program. When run, this program prints out the XML file to the screen.

XSU Generating XML Example 2: Generating DOM From Table emp (Java)

DOM (Document Object Model) is a standard defined by the W3C committee. DOM represents an XML document in a parsed tree-like form. Each XML entity becomes a DOM node. Thus XML elements and attributes become DOM nodes while their children become child nodes. To generate a DOM tree from the XML generated by XSU, you can directly request a DOM document from XSU, as it saves the overhead of having to create a string representation of the XML document and then parse it to generate the DOM tree.

XSU calls the parser to directly construct the DOM tree from the data values. The following example illustrates how to generate a DOM tree. The example steps through the DOM tree and prints all the nodes one by one.

```
import org.w3c.dom.*;
import oracle.xml.parser.v2.*;
import java.sql.*;
import oracle.xml.sql.query.OracleXMLQuery;
import java.io.*;

 class domTest{

   public static void main(String[] argv)
   {
      try{
      // create the connection
      Connection conn  = getConnection("scott","tiger");

      // Create the query class.
      OracleXMLQuery qry = new OracleXMLQuery(conn, "select * from emp");

      // Get the XML DOM object. The actual type is the Oracle Parser's DOM
      // representation. (XMLDocument)
      XMLDocument domDoc = (XMLDocument)qry.getXMLDOM();

      // Print the XML output directly from the DOM
      domDoc.print(System.out);

      // If you instead want to print it to a string buffer you can do
this..!
      StringWriter s = new StringWriter(10000);
      domDoc.print(new PrintWriter(s));
      System.out.println(" The string version ---> "+s.toString());

      qry.close(); // You should always close the query!!
      }catch(Exception e){
        System.out.println(e.toString());
      }
    }

    // Get the connection given the user name and password..!
    private static Connection getConnection(String user, String passwd)
      throws SQLException
    {
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
```

```
        Connection conn =
            DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
        return conn;
    }
}
```

Paginating Results: skipRows and maxRows

In the examples shown so far, XML SQL Utility (XSU) takes the ResultSet or the query and generates the whole document from all the rows of the query. To obtain 100 rows at a time, you would then have to fire off different queries to get the first 100 rows, the next 100, and so on. Also it is not possible to skip the first five rows of the query and then generate the result.

To obtain the desired results, use the XSU `skipRows` and `maxRows` parameter settings:

- `skipRows` parameter, when set, forces the generation to skip the desired number of rows before starting to generate the result.
- `maxRows` limits the number of rows converted to XML.

For example, if you set `skipRows` to a value of 5 and `maxRows` to a value of 10, then XSU skips the first 5 rows, then generates XML for the next 10 rows.

Keeping the Object Open for the Duration of the User's Session

In Web scenarios, you may want to keep the query object open for the duration of the user's session. For example, consider the case of a Web search engine which gives the results of a user's search in a paginated fashion. The first page lists 10 results, the next page lists 10 more results, and so on.

To achieve this, request XSU to convert 10 rows at a time and keep the ResultSet state active, so that the next time you ask XSU for more results, it starts generating from the place the last generation finished. See "XSU Generating XML Example 3: Paginating Results: Generating an XML Page (Java)" on page 8-30.

When the Number of Rows or Columns in a Row Is Too Large

There is also the case when the number of rows, or number of columns in a row are very large. In this case, you can generate multiple documents each of a smaller size.

These cases can be handled by using the `maxRows` parameter and the `keepObjectOpen` function.

keepObjectOpen Function

Typically, as soon as all results are generated, `OracleXMLQuery` internally closes the `ResultSet`, if it created one using the SQL query string given, since it assumes you no longer want any more results. However, in the case described earlier, to maintain that state, you need to call the `keepObjectOpen` function to keep the cursor active. See the following example.

XSU Generating XML Example 3: Paginating Results: Generating an XML Page (Java)

This example, writes a simple class that maintains the state and generates the next page each time it is called.

```
import org.w3c.dom.*;
import oracle.xml.parser.v2.*;
import java.sql.*;
import oracle.xml.sql.query.OracleXMLQuery;
import java.io.*;
public class pageTest
{
   Connection conn;
   OracleXMLQuery qry;
   ResultSet rset;
   Statement stmt;
   int lastRow = 0;

   public pageTest(String sqlQuery)
   {
     try{
     conn  = getConnection("scott","tiger");
     //stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
     //             ResultSet.CONCUR_READ_ONLY);// create a scrollable Rset
     //stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
     //             ResultSet.CONCUR_READ_ONLY);// create a scrollable Rset
     stmt = conn.createStatement();
     ResultSet rset = stmt.executeQuery(sqlQuery);  // get the result set..
     rset.first();
     qry = new OracleXMLQuery(conn,rset);   // create a OracleXMLQuery instance
     qry.keepCursorState(true); // Don't lose state after the first fetch
     qry.setRaiseNoRowsException(true);
     qry.setRaiseException(true);
```

```
  }catch(SQLException e){
   System.out.println(e.toString());
  }
}

// Returns the next XML page..!
public String getResult(int startRow, int endRow)  throws SQLException
{
  //rset.relative(lastRow-startRow);  // scroll inside the result set
  //rset.absolute(startRow);  // scroll inside the result set
  qry.setMaxRows(endRow-startRow); // set the max # of rows to retrieve..!
  //System.out.println("before getxml");
  return qry.getXMLString();
}

// Function to still perform the next page.
public String nextPage() throws SQLException
{
  String result = getResult(lastRow,lastRow+10);
  lastRow+= 10;
  return result;
}

public void close() throws SQLException
{
  stmt.close();   // close the statement..
  conn.close();   // close the connection
  qry.close();    // close the query..
}

public static void main(String[] argv)
{
  String str;

  try{
  pageTest test = new pageTest("select e.* from emp e");

  int i = 0;
  // Get the data one page at a time..!!!!!
  while ((str = test.getResult(i,i+10))!= null)
  {
      System.out.println(str);
      i+= 10;
  }
  test.close();
```

```
    }catch(Exception e){
     e.printStackTrace(System.out);
    }
   }
 // Get the connection given the user name and password..!
    private static Connection getConnection(String user, String passwd)
      throws SQLException
    {
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
      Connection conn =
          DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
     return conn;
   }

}
```

Generating XML from ResultSet Objects

You saw how you can supply a SQL query and get the results as XML. In the last example, you retrieved paginated results. However in Web cases, you may want to retrieve the previous page and not just the next page of results. To provide this scrollable functionality, you can use the Scrollable `ResultSet`. Use the `ResultSet` object to move back and forth within the result set and use XSU to generate the XML each time. The following example illustrates how to do this.

XSU Generating XML Example 4: Generating XML from JDBC ResultSets (Java)

This example shows you how to use the JDBC `ResultSet` to generate XML. Note that using the `ResultSet` might be necessary in cases that are not handled directly by XSU, for example, when setting the batch size, binding values, and so on. This example extends the previously defined `pageTest` class to handle any page.

```
public class pageTest()
{
   Connection conn;
   OracleXMLQuery qry;
   ResultSet rset;
   int lastRow = 0;

   public pageTest(String sqlQuery)
   {
     conn  = getConnection("scott","tiger");
     Statement stmt = conn.createStatement(sqlQuery);// create a scrollable Rset
     ResultSet rset = stmt.executeQuery();  // get the result set..
```

```
      qry = new OracleXMLQuery(conn,rset);   // create a OracleXMLQuery instance
      qry.keepObjectOpen(true); // Don't lose state after the first fetch
    }

    // Returns the next XML page..!
    public String getResult(int startRow, int endRow)
    {
      rset.scroll(lastRow-startRow);  // scroll inside the result set
      qry.setMaxRows(endRow-startRow); // set the max # of rows to retrieve..!
      return qry.getXMLString();
    }

    // Function to still perform the next page.
    public String nextPage()
    {
      String result = getResult(lastRow,lastRow+10);
      lastRow+= 10;
      return result;
    }

    public void close()
    {
      stmt.close();   // close the statement..
      conn.close();   // close the connection
      qry.close();    // close the query..
    }

    public void main(String[] argv)
    {
      pageTest test = new pageTest("select * from emp");

      int i = 0;
      // Get the data one page at a time..!!!!!
      while ((str = test.getResult(i,i+10))!= null)
      {
          System.out.println(str);
          i+= 10;
      }
      test.close();
    }
}
```

XSU Generating XML Example 5: Generating XML from Procedure Return Values

The `OracleXMLQuery` class provides XML conversion only for query strings or `ResultSets`. But in your application if you have PL/SQL procedures that return REF cursors, how would you do the conversion?

In this case, you can use the earlier-mentioned ResultSet conversion mechanism to perform the task. REF cursors are references to cursor objects in PL/SQL. These cursor objects are valid SQL statements that can be iterated upon to get a set of values. These REF cursors are converted into `OracleResultSet` objects in the Java world.

You can execute these procedures, get the `OracleResultSet` object, and then send that to the `OracleXMLQuery` object to get the desired XML.

Consider the following PL/SQL function that defines a REF cursor and returns it:

```
CREATE OR REPLACE package body testRef is

  function testRefCur RETURN empREF is
  a empREF;
  begin
      OPEN a FOR select * from scott.emp;
      return a;
  end;
end;
/
```

Every time this function is called, it opens a cursor object for the query, `select * from emp` and returns that cursor instance. To convert this to XML, you can do the following:

```
import org.w3c.dom.*;
import oracle.xml.parser.v2.*;
import java.sql.*;
import oracle.jdbc.driver.*;
import oracle.xml.sql.query.OracleXMLQuery;
import java.io.*;
public class REFCURtest
{
   public static void main(String[] argv)
     throws SQLException
   {
      String str;
      Connection conn  = getConnection("scott","tiger"); // create connection
```

```
        // Create a ResultSet object by calling the PL/SQL function
        CallableStatement stmt =
           conn.prepareCall("begin ? := testRef.testRefCur(); end;");

        stmt.registerOutParameter(1,OracleTypes.CURSOR); // set the define type

        stmt.execute();   // Execute the statement.
        ResultSet rset = (ResultSet)stmt.getObject(1);  // Get the ResultSet

        OracleXMLQuery qry = new OracleXMLQuery(conn,rset); // prepare Query class
        qry.setRaiseNoRowsException(true);
        qry.setRaiseException(true);
        qry.keepCursorState(true);        // set options (keep the cursor active.
        while ((str = qry.getXMLString())!= null)
             System.out.println(str);

        qry.close();    // close the query..!

        // Note since we supplied the statement and resultset, closing the
        // OracleXMLquery instance will not close these. We would need to
        // explicitly close this ourselves..!
        stmt.close();
        conn.close();
      }
       // Get the connection given the user name and password..!
       private static Connection getConnection(String user, String passwd)
         throws SQLException
       {
         DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
         Connection conn =
             DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
        return conn;
      }

}
```

To apply the stylesheet, on the other hand, use the `applyStylesheet()` command. This forces the stylesheet to be applied before generating the output.

Raising No Rows Exception

When there are no rows to process, XSU simply returns a null string. However, it might be desirable to get an exception every time there are no more rows present, so that the application can process this through exception handlers. When the

`setRaiseNoRowsException()` is set, then whenever there are no rows to generate for the output XSU raises an `oracle.xml.sql.OracleXMLSQLNoRowsException`. This is a run time exception and need not be caught unless needed.

XSU Generating XML Example 6: No Rows Exception (Java)

The following code extends the previous examples to use the exception instead of checking for null strings:

```
public class pageTest {
    .... // rest of the class definitions....

   public void main(String[] argv)
   {
     pageTest test = new pageTest("select * from emp");

     test.query.setRaiseNoRowsException(true); // ask it to generate
exceptions
     try
     {
        while(true)
         System.out.println(test.nextPage());
     }
     catch(oracle.xml.sql.OracleXMLNoRowsException)
     {
       System.out.println(" END OF OUTPUT ");
       test.close();
     }
   }
}
```

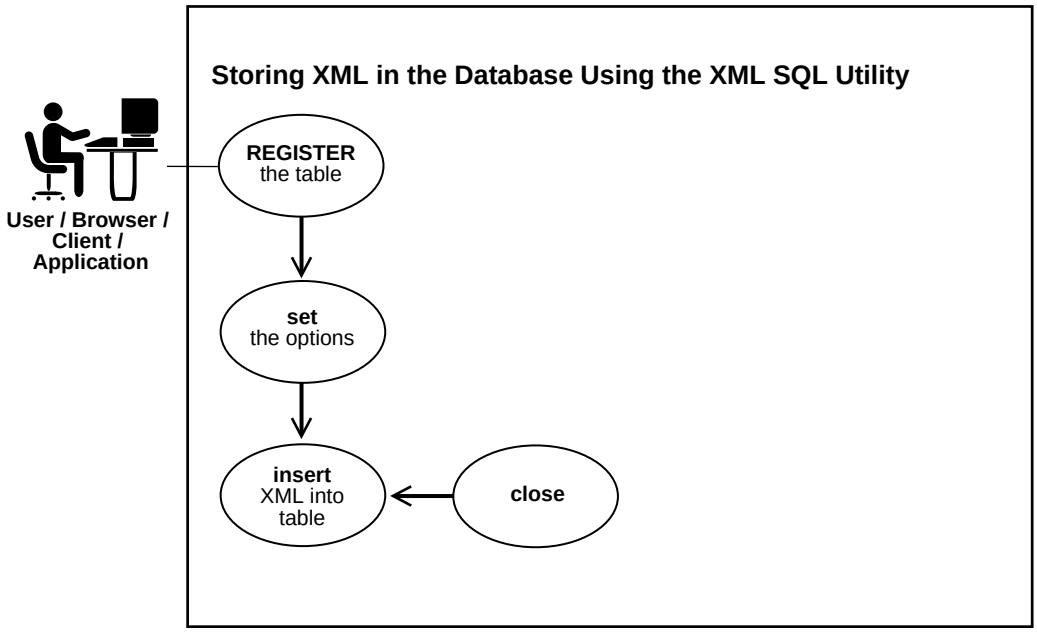
Note: Notice how the condition to check the termination changed from checking for the result to be `NULL` to an exception handler.

Storing XML Back in the Database Using XSU OracleXMLSave

Now that you have seen how queries can be converted to XML, observe how you can put the XML back into the tables or views using XSU. The class `oracle.xml.sql.dml.OracleXMLSave` provides this functionality. It has methods to insert XML into tables, update existing tables with the XML document, and delete rows from the table based on XML element values.

In all these cases the given XML document is parsed, and the elements are examined to match tag names to column names in the target table or view. The elements are converted to the SQL types and then bound to the appropriate statement. The process for storing XML using XSU is shown in Figure 8–6.

Figure 8–6 Storing XML in the Database Using XML SQL Utility



Consider an XML document that contains a list of ROW elements, each of which constitutes a separate DML operation, namely, `insert`, `update`, or `delete` on the table or view.

Insert Processing Using XSU (Java API)

To insert a document into a table or view, simply supply the table or the view name and then the document. XSU parses the document (if a string is given) and then creates an INSERT statement into which it binds all the values. By default, XSU inserts values into all the columns of the table or view and an absent element is treated as a NULL value. The following example shows you how the XML document generated from the emp table, can be stored in the table with relative ease.

XSU Inserting XML Example 7: Inserting XML Values into All Columns (Java)

This example inserts XML values into all columns:

```
// This program takes as an argument the file name, or a url to
// a properly formated XML document and inserts it into the SCOTT.EMP table.
import java.sql.*;
import oracle.xml.sql.dml.OracleXMLSave;
public class testInsert
{
   public static void main(String argv[])
     throws SQLException
  {
     DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
     Connection conn =
          DriverManager.getConnection("jdbc:oracle:oci8:@","scott","tiger");

     OracleXMLSave sav = new OracleXMLSave(conn, "emp");
     sav.insertXML(sav.getUrl(argv[0]));
     sav.close();
   }
}
```

An INSERT statement of the form:

```
insert into scott.emp (EMPNO, ENAME, JOB, MGR, SAL, DEPTNO) VALUES(?,?,?,?,?,?);
```

is generated, and the element tags in the input XML document matching the column names are matched and their values bound.

If you store the following XML document:

```
 <?xml version='1.0'?>
<ROWSET>
  <ROW num="1">
    <EMPNO>7369</EMPNO>
    <ENAME>Smith</ENAME>
```

```
    <JOB>CLERK</JOB>
    <MGR>7902</MGR>
    <HIREDATE>12/17/1980 0:0:0</HIREDATE>
    <SAL>800</SAL>
    <DEPTNO>20</DEPTNO>
  </ROW>
  <!-- additional rows ... -->
</ROWSET>
```

to a file and specify the file to the program described earlier, you would end up with a new row in the `emp` table containing the values (`7369, Smith, CLERK, 7902, 12/17/1980,800,20`). Any element absent inside the row element is taken as a null value.

XSU Inserting XML Example 8: Inserting XML Values into Columns (Java)

In certain cases, you may not want to insert values into *all* columns. This may be true when the group of values that you are getting is not the complete set and you need triggers or default values to be used for the rest of the columns. The code following shows how this can be done.

Assume that you are getting the values only for the employee number, name, and job and that the salary, manager, department number, and hire date fields are filled in automatically. First create a list of column names that you want the `insert` to work on and then pass it to the `OracleXMLSave` instance.

```
import java.sql.*;
import oracle.xml.sql.dml.OracleXMLSave;
public class testInsert
{
   public static void main(String argv[])
     throws SQLException
   {
      Connection conn = getConnection("scott","tiger");
      OracleXMLSave sav = new OracleXMLSave(conn, "scott.emp");

      String [] colNames = new String[5];
      colNames[0] = "EMPNO";
      colNames[1] = "ENAME";
      colNames[2] = "JOB";

      sav.setUpdateColumnList(colNames); // set the columns to update..!

      // Assume that the user passes in this document as the first argument!
```

```
      sav.insertXML(argv[0]);
      sav.close();
   }
   // Get the connection given the user name and password..!
    private static Connection getConnection(String user, String passwd)
      throws SQLException
    {
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
      Connection conn =
          DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
     return conn;
   }
}
```

An `insert` statement of the form:

```
insert into scott.emp (EMPNO, ENAME, JOB) VALUES (?, ?, ?);
```

is generated. Note that, in the preceding example, if the inserted document contains values for the other columns (`JOB, HIREDATE`, and so on), those are ignored. Also an `insert` is performed for each `ROW` element that is present in the input. These inserts are batched by default.

Update Processing Using XSU (Java API)

Now that you know how to insert values into the table from XML documents, see how you can update only *certain* values. In an XML document, to update the salary of an employee and the department that they work in:

```
<ROWSET>
  <ROW num="1">
    <EMPNO>7369</EMPNO>
    <SAL>1800</SAL>
    <DEPTNO>30</DEPTNO>
  </ROW>
  <ROW>
    <EMPNO>2290</EMPNO>
    <SAL>2000</SAL>
    <HIREDATE>12/31/1992</HIREDATE>
  <!-- additional rows ... -->
</ROWSET>
```

You can use the XSU to update the values. For updates, you must supply XSU with the list of key column names. These form part of the `WHERE` clause in the `UPDATE`

statement. In the emp table shown earlier, employee number (EMPNO) column forms the key. Use this for updates.

XSU Updating XML Example 9: Updating a Table Using the keyColumns (Java)

This example updates table, emp, using keyColumns:

```
import java.sql.*;
import oracle.xml.sql.dml.OracleXMLSave;
public class testUpdate
{
   public static void main(String argv[])
     throws SQLException
   {
      Connection conn = getConnection("scott","tiger");
      OracleXMLSave sav = new OracleXMLSave(conn, "scott.emp");

      String [] keyColNames = new String[1];
      keyColNames[0] = "EMPNO";
      sav.setKeyColumnList(keyColNames);

      // Assume that the user passes in this document as the first argument!
      sav.updateXML(argv[0]);
      sav.close();
   }
   // Get the connection given the user name and password..!
    private static Connection getConnection(String user, String passwd)
      throws SQLException
    {
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
      Connection conn =
          DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
     return conn;
   }
}
```

In this example, two UPDATE statements are generated. For the first ROW element, you generate an UPDATE statement to update the SAL and JOB fields as follows:

```
update scott.emp SET SAL = 1800 and DEPTNO = 30 WHERE EMPNO = 7369;
```

For the second ROW element:

```
update scott.emp SET SAL = 2000 and HIREDATE = 12/31/1992 WHERE EMPNO = 2290;
```

XSU Updating XML Example 10: Updating a Specified List of Columns (Java)

You may want to specify a *list* of columns to update. This would speed up the processing since the same UPDATE statement can be used for all the ROW elements. Also you can ignore other tags in the XML document.

> **Note:** When you specify a list of columns to update, an element corresponding to one of the update columns, if absent, will be treated as NULL.

If you know that all the elements to be updated are the same for all the ROW elements in the XML document, you can use the `setUpdateColumnNames()` function to set the list of columns to update.

```
import java.sql.*;
import oracle.xml.sql.dml.OracleXMLSave;
public class testUpdate
{
   public static void main(String argv[])
     throws SQLException
   {
      Connection conn = getConnection("scott","tiger");
      OracleXMLSave sav = new OracleXMLSave(conn, "scott.emp");

      String [] keyColNames = new String[1];
      keyColNames[0] = "EMPNO";
      sav.setKeyColumnList(keyColNames);

      // you create the list of columns to update..!
      // Note that if you do not supply this, then for each ROW element in the
      // XML document, you would generate a new update statement to update all
      // the tag values (other than the key columns)present in that element.
      String[] updateColNames = new String[2];
      updateColNames[0] = "SAL";
      updateColNames[1] = "JOB";
      sav.setUpdateColumnList(updateColNames); // set the columns to update..!

      // Assume that the user passes in this document as the first argument!
      sav.updateXML(argv[0]);
      sav.close();
   }
   // Get the connection given the user name and password..!
    private static Connection getConnection(String user, String passwd)
```

```
    throws SQLException
  {
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    Connection conn =
        DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
   return conn;
  }
}
```

Delete Processing Using XSU (Java API)

When deleting from XML documents, you can set the list of key columns. These columns are used in the WHERE clause of the DELETE statement. If the key column names are not supplied, then a new DELETE statement is created for each ROW element of the XML document, where the list of columns in the WHERE clause of the DELETE statement will match those in the ROW element.

XSU Deleting XML Example 11: Deleting Operations Per Row (Java)

Consider this delete example:

```
import java.sql.*;
import oracle.xml.sql.dml.OracleXMLSave;
public class testDelete
{
   public static void main(String argv[])
     throws SQLException
   {
      Connection conn = getConnection("scott","tiger");
      OracleXMLSave sav = new OracleXMLSave(conn, "scott.emp");

      // Assume that the user passes in this document as the first argument!
      sav.deleteXML(argv[0]);
      sav.close();
   }
   // Get the connection given the user name and password..!
    private static Connection getConnection(String user, String passwd)
      throws SQLException
    {
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
      Connection conn =
          DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
     return conn;
   }
```

```
}
```

Using the same XML document shown previously for the update example, you would end up with two DELETE statements:

```
DELETE FROM scott.emp WHERE empno=7369 and sal=1800 and deptno=30;
DELETE FROM scott.emp WHERE empno=2200 and sal=2000 and hiredate=12/31/1992;
```

The DELETE statements were formed based on the tag names present in each ROW element in the XML document.

XSU Deleting XML Example 12: Deleting Specified Key Values (Java)

If instead, you want the DELETE statement to only use the key values as predicates, you can use the setKeyColumn function to set this.

```
import java.sql.*;
import oracle.xml.sql.dml.OracleXMLSave;
public class testDelete
{
   public static void main(String argv[])
     throws SQLException
   {
      Connection conn = getConnection("scott","tiger");
      OracleXMLSave sav = new OracleXMLSave(conn, "scott.emp");

      String [] keyColNames = new String[1];
      keyColNames[0] = "EMPNO";
      sav.setKeyColumnList(keyColNames);

      // Assume that the user passes in this document as the first argument!
      sav.deleteXML(argv[0]);
      sav.close();
   }
   // Get the connection given the user name and password..!
    private static Connection getConnection(String user, String passwd)
      throws SQLException
    {
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
      Connection conn =
          DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
     return conn;
   }
}
```

Here is a single DELETE statement of the form:

```
DELETE FROM scott.emp WHERE EMPNO=?
```

Advanced XSU Usage Techniques

XSU Exception Handling in Java

OracleXMLSQLException class

XSU catches all exceptions that occur during processing and throws an `oracle.xml.sql.OracleXMLSQLException` which is a run time exception. The calling program thus does not have to catch this exception all the time, if the program can still catch this exception and do the appropriate action. The exception class provides functions to get the error message and also get the parent exception, if any. For example, the program shown later, catches the run time exception and then gets the parent exception.

OracleXMLNoRowsException class

This exception is generated when the `setRaiseNoRowsException` is set in the `OracleXMLQuery` class during generation. This is a subclass of the `OracleXMLSQLException` class and can be used as an indicator of the end of row processing during generation.

```
import java.sql.*;
import oracle.xml.sql.query.OracleXMLQuery;

public class testException
{
   public static void main(String argv[])
     throws SQLException
   {
      Connection conn = getConnection("scott","tiger");

      // wrong query this will generate an exception
      OracleXMLQuery qry = new OracleXMLQuery(conn, "select * from emp where sd
 = 322323");

      qry.setRaiseException(true); // ask it to raise exceptions..!

      try{
        String str = qry.getXMLString();
```

```
      }catch(oracle.xml.sql.OracleXMLSQLException e)
      {
        // Get the original exception
        Exception parent = e.getParentException();
        if (parent instanceof java.sql.SQLException)
        {
           // perform some other stuff. Here you simply print it out..
           System.out.println(" Caught SQL Exception:"+parent.getMessage());
        }
        else
          System.out.println(" Exception caught..!"+e.getMessage());
     }
   }
    // Get the connection given the user name and password..!
    private static Connection getConnection(String user, String passwd)
      throws SQLException
    {
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
      Connection conn =
          DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
     return conn;
   }
}
```

Frequently Asked Questions About XML SQL Utility (XSU)

This section lists XML SQL Utility (XSU) questions and answers.

What Schema Structure Should I Use with XSU to Store XML?

I have the following XML in my `customer.xml` file:

```
<ROWSET>
 <ROW num="1">
  <CUSTOMER>
   <CUSTOMERID>1044</CUSTOMERID>
   <FIRSTNAME>Paul</FIRSTNAME>
   <LASTNAME>Astoria</LASTNAME>
   <HOMEADDRESS>
    <STREET>123 Cherry Lane</STREET>
    <CITY>SF</CITY>
    <STATE>CA</STATE>
    <ZIP>94132</ZIP>
   </HOMEADDRESS>
```

```
  </CUSTOMER>
 </ROW>
</ROWSET>
```

What database schema structure should I use to store this XML with XSU?

Answer: Since your example is more than one level deep (that is, it has a nested structure), you should use an object-relational schema. The XML preceding will canonically map to such a schema. An appropriate database schema would be the following:

```
create type address_type as object
 (
 street varchar2(40),
 city varchar2(20),
 state varchar2(10),
 zip varchar2(10)
 );
 /
 create type customer_type as object
 (
customerid number(10),
firstname varchar2(20),
lastname varchar2(20),
homeaddress address_type
 );
/
create table customer_tab ( customer customer_type);
```

In the case you wanted to load `customer.xml` by means of XSU into a relational schema, you can still do it by creating objects in views on top of your relational schema.

For example, you would have a relational table which would contain all the following information:

```
create table cust_tab
 ( customerid number(10),
   firstname varchar2(20),
   lastname varchar2(20),
   state varchar2(40),
   city varchar2(20),
   state varchar2(20),
   zip varchar2(20)
 );
```

Then, you would create a customer view which contains a customer object on top of it, as in the following example:

```
create view customer_view as
select customer_type(customerid, firstname, lastname,
address_type(state,street,city,zip))
from cust_tab;
```

Finally, you can flatten your XML using XSLT and then insert it directly into your relational schema. However, this is the least recommended option.

Can XSU Store XML Data Across Tables?

Answer: Currently the XML SQL Utility (XSU) can only store data in a single table. It maps a canonical representation of an XML document into any table or view. But there is a way to store XML with XSU across tables. One can do this using XSLT to transform any document into multiple documents and insert them separately. Another way is to define views over multiple tables (using object views if needed) and then do the `inserts` into the view. If the view is inherently non-updatable (because of complex joins), then you can use `INSTEAD OF` triggers over the views to do the inserts.

Can I Use XSU to Load XML Stored in Attributes?

I would like to use XSU to load XML where some of the data is stored in attributes. However, XSU seems to ignore the XML attributes. What can I do?

Answer: Unfortunately, for now you will have to use XSLT to transform your XML document; that is, you must change the attributes into elements. XSU does assume canonical mapping from XML to a database schema. This takes away a bit from the flexibility, forcing you to sometimes resort to XSLT, but at the same time, in the common case, it does not burden you with having to specify a mapping.

Is XSU Case-Sensitive? Can I Use ignoreCase?

I am trying to insert the following XML document (`dual.xml`):

```
<ROWSET>
   <row>
      <DUMMY>X</DUMMY>
   </row>
</ROWSET>
```

Into the table dual using the command line front end of the XSU, like in this example:

```
java OracleXML putxml -filename dual.xml dual
```

I get the following error:

```
oracle.xml.sql.OracleXMLSQLException: No rows to modify -- the row enclosing tag
missing.  Specify the correct row enclosing tag.
```

Answer: By default, XSU is case sensitive, so it looks for the record separator tag which by default is ROW, yet all it can find is row. Another common, related mistake is to mismatch the case of one of the element tags. For example, if in dual.xml the tag DUMMY was actually dummy, then XSU raises an error stating that it could not find a matching column in table, dual. So you have two options: use the correct case or use the ignoreCase feature.

Will XSU Generate the Database Schema from a DTD?

Answer: No. Due to a number of shortcomings of the DTD, this functionality is not available. The W3C XML Schema recommendation is finalized, but this functionality is not available yet in XSU.

Can You Provide a Thin Driver Connect String Example for XSU?

I am using the XML SQL Utility command line front end, and I am passing a connect string but I get a TNS error. Can you provide examples of a thin driver connect string and an OCI8 driver connect string?

Answer: An example of an JDBC thin driver connect string is:

```
jdbc:oracle:thin:<user>/<password>@<hostname>:<port number>:<DB SID>;
```

Furthermore, the database must have an active TCP/IP listener. A valid OCI8 connect string would be:

```
jdbc:oracle:oci8:<user>/<password>@<hostname>
```

Does XSU Commit After INSERT, DELETE, or UPDATE?

Does XML SQL Utility commit after it is done inserting, deleting, or updating? What happens if an error occurs?

Answer: By default the XSU executes a number of `insert`, `delete`, or `update` statements at a time. The number of statements batch together and executed at the same time can be overridden using the `setBatchSize` feature.

Also, by default XSU does no explicit commits. If autocommit is on (default for the JDBC connection), then after each batch of statement executions a commit occurs. You can override this by turning autocommit off and then specifying after how many statement executions a commit should occur, which can be done using the `setCommitBatch` feature.

If an error occurs, XSU rolls back to either the state the target table was in before the particular call to XSU, or the state right after the last commit made during the current call to XSU.

Can You Explain How to Map Table Columns to XML Attributes Using XSU?

Can you explain how to map table columns to XML attributes using XSU?

Answer: From XSU release 2.1.0 you can map a particular column or a group of columns to an XML attribute instead of an XML element. To achieve this, you have to create an alias for the column name, and prepend the at sign (@) to the name of this alias. For example:

```
* Create a file called select.sql with the following content :
   SELECT empno "@EMPNO", ename, job, hiredate
   FROM emp
   ORDER BY empno

 * Call the XML SQL Utility :
   java OracleXML getXML -user "scott/tiger" \
           -conn "jdbc:oracle:thin:@myhost:1521:ORCL" \
           -fileName "select.sql"

 * As a result, the XML document will look like :
     <?xml version = '1.0'?>
     <ROWSET>
        <ROW num="1" EMPNO="7369">
           <ENAME>SMITH</ENAME>
           <JOB>CLERK</JOB>
           <HIREDATE>12/17/1980 0:0:0</HIREDATE>
        </ROW>
        <ROW num="2" EMPNO="7499">
           <ENAME>ALLEN</ENAME>
           <JOB>SALESMAN</JOB>
           <HIREDATE>2/20/1981 0:0:0</HIREDATE>
```

```
    </ROW>
</ROWSET>
```

Note: All attributes must appear *before* any non-attribute.

Since the XML document is created in a streamed manner, the following query:

```
SELECT ename, empno "@EMPNO", ...
```

would not generate the expected result. It is currently not possible to load XML data stored in attributes. You will still need to use an XSLT transformation to change the attributes into elements. XSU assumes canonical mapping from XML to a database schema.

9

XSQL Pages Publishing Framework

This chapter contains the following sections:

XSQL Pages Publishing Framework Overview

The Oracle XSQL Pages publishing framework is an extensible platform for easily publishing XML information in any format you desire. It greatly simplifies combining the power of SQL, XML, and XSLT to publish dynamic web content based on database information.

Using the XSQL publishing framework, anyone familiar with SQL can create and use declarative templates called "XSQL pages" to:

- Assemble dynamic XML "datagrams" based on parameterized SQL queries, and
- Transform these "data pages" to produce a final result in any desired XML, HTML, or text-based format using an associated XSLT transformation.

Assembling and transforming information for publishing requires no programming. In fact, most of the common things you will want to do can be easily achieved in a declarative way. However, since the XSQL publishing framework is extensible, if one of the built-in features does not fit your needs, you can easily extend the framework using Java to integrate custom information sources or to perform custom server-side processing.

Using the XSQL Pages framework, the *assembly* of information to be published is cleanly separated from presentation. This simple architectural detail has profound productivity benefits. It allows you to:

- Present the same information in multiple ways, including tailoring the presentation appropriately to the kind of client device making the request (browser, cellular phone, PDA, and so on).
- Reuse information easily by aggregating existing pages into new ones
- Revise and enhance the presentation independently of the information content being presented.

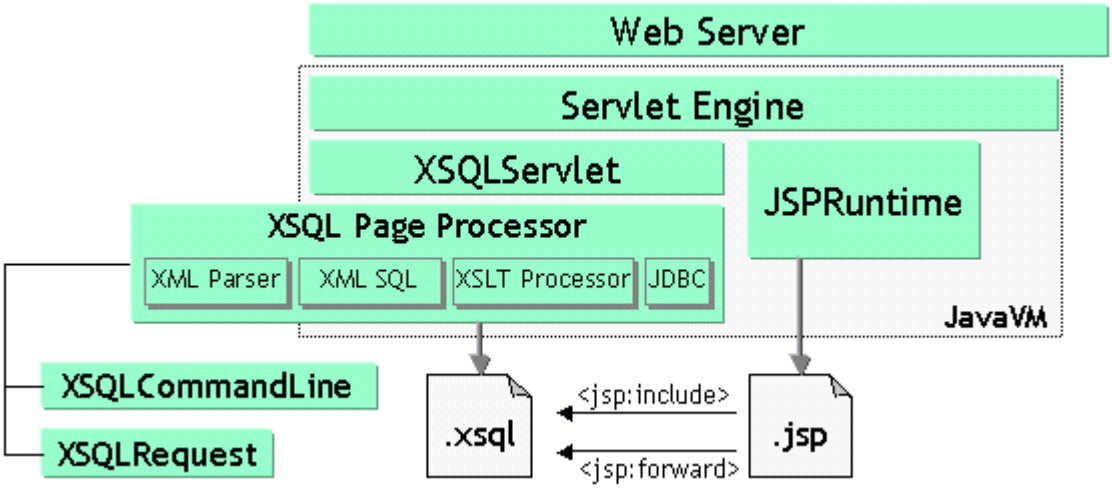
What Can I Do with Oracle XSQL Pages?

Using server-side templates — known as "XSQL pages" due to their `.xsql` extension — you can publish any information in any format to any device. The XSQL page processor "engine" interprets, caches, and processes the contents of your XSQL page templates. Figure 9–1 illustrates that the core XSQL page processor engine can be "exercised" in four different ways:

- From the command line or in batch using the *XSQL Command-Line Utility*
- Over the Web, using the *XSQL Servlet* installed into your favorite web server

- As part of JSP applications, using `<jsp:include>` to include a template
- Programmatically, with the `XSQLRequest` object, the engine's Java API

Figure 9–1 Understanding the Architecture of the XSQL Pages Framework



The same XSQL page templates can be used in any or all of these scenarios. Regardless of the means by which a template is processed, the same basic steps occur to produce a result. The XSQL page processor "engine":

1. Receives a request to process an XSQL template
2. Assembles an XML "datagram" using the result of one or more SQL queries
3. Returns this XML "datagram" to the requestor
4. Optionally transforms the "datagram" into any XML, HTML, or text format

During the transformation step in this process, you can use stylesheets that conform to the W3C XSLT 1.0 standard to transform the assembled "datagram" into document formats like:

- HTML for browser display
- Wireless Markup Language (WML) for wireless devices
- Scalable Vector Graphics (SVG) for data-driven charts, graphs, and diagrams
- XML Stylesheet Formatting Objects (XSL-FO), for rendering into Adobe PDF

- Text documents, like emails, SQL scripts, Java programs, and so on.
- Arbitrary XML-based document formats

XSQL Pages bring this functionality to you by automating the use of underlying Oracle XML components to solve many common cases without resorting to custom programming. However, when only custom programming will do — as we'll see in the Advanced Topics section of this chapter — you can augment the framework's built-in actions and serializers to assemble the XSQL "datagrams" from any custom source and serialize the datagrams into any desired format, without having to write an entire publishing framework from scratch.

See Also:

- Chapter A, "XDK for Java: Specifications and Quick References" for the XSQL Servlet specifications and cheat sheets.
- XSQL Servlet Release Notes on OTN at: http://otn.oracle.com/tech/xml

Where Can I Obtain Oracle XSQL Pages?

XSQL Servlet is provided with Oracle9*i* and *is* also available for download from the OTN site: http://otn.oracle.com/tech/xml.

Where indicated, the examples and demos described in this chapter are also available from OTN.

What's Needed to Run XSQL Pages?

To run the Oracle XSQL Pages publishing framework from the command-line, all you need is a Java VM (1.1.8, 1.2.2, or 1.3). The XSQL Pages framework depends on two underlying components in the Oracle XML Developer's Kit:

- Oracle XML Parser and XSLT Processor (`xmlparserv2.jar`)
- Oracle XML SQL Utility (`xsu12.jar`)

Both of their Java archive files must be present in the CLASSPATH where the XSQL pages framework is running. Since most XSQL pages will connect to a database to query information for publishing, the framework also depends on a JDBC driver. Any JDBC driver is supported, but when connecting to Oracle, it's best to use the Oracle JDBC driver (`classes12.jar`) for maximum functionality and performance.

Lastly, the XSQL publishing engine expects to read its configuration file named `XSQLConfig.xml` as a Java resource, so you must include the *directory* where the `XSQLConfig.xml` file resides in the CLASSPATH as well.

To use the XSQL Pages framework for Web publishing, in addition to the preceding you need a web server that supports Java Servlets. The following is the list of web servers with Servlet capability on which the XSQL Servlet has been tested:

- Oracle9*i* Internet Application Server v1.x and v2.x
- Oracle9*i* Oracle Servlet Engine
- Allaire JRun 2.3.3 and 3.0.0
- Apache 1.3.9 or higher with JServ 1.0/1.1 or Tomcat 3.1/3.2 Servlet Engine
- Apache Tomcat 3.1 or 3.2 Web Server + Servlet Engine
- Caucho Resin 1.1
- Java Web Server 2.0
- Weblogic 5.1 Web Server
- NewAtlanta ServletExec 2.2 and 3.0 for IIS/PWS 4.0
- Oracle8*i* Lite Web-to-Go Server
- Sun JavaServer Web Development Kit (JSWDK) 1.0.1 Web Server

---

Note: For security reasons, when installing XSQL Servlet on your production web server, make sure `XSQLConfig.xml` file does *not* reside in a directory that is part of the web server's virtual directory hierarchy. Failure to take this precaution risks exposing your configuration information over the web.

For details on installing, configuring your environment, and running XSQL Servlet and for additional examples and guidelines, see the XSQL Servlet "Release Notes" on OTN at `http://otn.oracle.com/tech/xml`

Overview of Basic XSQL Pages Features

In this section, we'll get take a brief look at the most basic features you can exploit in your server-side XSQL page templates:

- Producing XML Datagrams from SQL Queries

- Transforming the XML Datagram into an Alternative XML Format
- Transforming the XML Datagram into HTML for Display

Producing XML Datagrams from SQL Queries

It is extremely easy to serve database information in XML format over the Web using XSQL pages. For example, let's see how simple it is to serve a real-time XML "datagram" from Oracle9*i*, of all available flights landing today at JFK airport. Using Oracle JDeveloper, or your favorite text editor, just build an XSQL page template like the one following, and save it in a file named, `AvailableFlightsToday.xsql`:

```
<?xml version="1.0"?>
<xsql:query connection="demo" bind-params="City" xmlns:xsql="urn:oracle-xsql">
    SELECT Carrier, FlightNumber, Origin, TO_CHAR(ExpectedTime,'HH24:MI') AS Due
      FROM FlightSchedule
     WHERE TRUNC(ExpectedTime) = TRUNC(SYSDATE) AND Arrived = 'N'
       AND Destination = ?   /* The ? is a bind variable being bound */
  ORDER BY ExpectedTime      /* to the value of the City parameter   */
</xsql:query>
```

With XSQL Servlet properly installed on your web server, you just need to copy the `AvailableFlightsToday.xsql` file preceding to a directory under your web server's virtual directory hierarchy. Then you can access the template through a web browser by requesting the URL:

```
http://yourcompany.com/AvailableFlightsToday.xsql?City=JFK
```

The results of the query in your XSQL page are materialized automatically as XML and returned to the requestor. This XML-based "datagram" would typically be requested by another server program for processing, but if you are using a browser such as Internet Explorer 5.0, you can directly view the XML result as shown in Figure 9–2.

Figure 9–2 XML Result From XSQL Page (AvailableFlightsToday.xsq) Query

Let's take a closer look at the "anatomy" of the XSQL page template we used. Notice the XSQL page begins with:

```
<?xml version="1.0"?>
```

This is because the XSQL template is itself an XML file (with an `*.xsql` extension) that contains any mix of static XML content and XSQL "action elements". The `AvailableFlightsToday.xsql` example preceding contains no *static* XML

elements, and just a single XSQL action element <xsql:query>. It represents the simplest useful XSQL page we can build, one that just contains a single query.

Notice that the first (and in this case, only!) element in the page <xsql:query> includes a special attribute that declares the xsql namespace prefix as a "synonym" for the Oracle XSQL namespace identifier urn:oracle-xsql.

```
<xsql:query connection="demo" bind-params="City" xmlns:xsql="urn:oracle-xsql">
```

This first, outermost element — known at the "document element" — also contains a connection attribute whose value "demo" is the name of one of the pre-defined connections in the XSQLConfig.xml configuration file:

```
<xsql:query connection="demo" bind-params="City" xmlns:xsql="urn:oracle-xsql">
```

The details concerning the username, password, database, and JDBC driver that will be used for the "demo" connection are centralized into the configuration file. Setting up these connection definitions is discussed in a later section of this chapter.

Lastly, the <xsql:query> element contains a bind-params attribute that associates the values of parameters in the request by name to bind parameters represented by question marks in the SQL statement contained inside the <xsql:query> tag.

Note that if we wanted to include more than one query on the page, we'll need to invent an XML element of our own creation to "wrap" the other elements like this:

```
<?xml version="1.0"?>
<page connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:query bind-params="City">
    SELECT Carrier, FlightNumber, Origin, TO_CHAR(ExpectedTime,'HH24:MI') AS Due
      FROM FlightSchedule
     WHERE TRUNC(ExpectedTime) = TRUNC(SYSDATE) AND Arrived = 'N'
       AND Destination = ?   /* The ? is a bind variable being bound */
      ORDER BY ExpectedTime  /* to the value of the City parameter   */
  </xsql:query>
  <!-- Other xsql:query actions can go here inside <page> and </page> -->
</page>
```

Notice in this example that the connection attribute and the xsql namespace declaration *always* go on the document element, while the bind-params is specific to the <xsql:query> action.

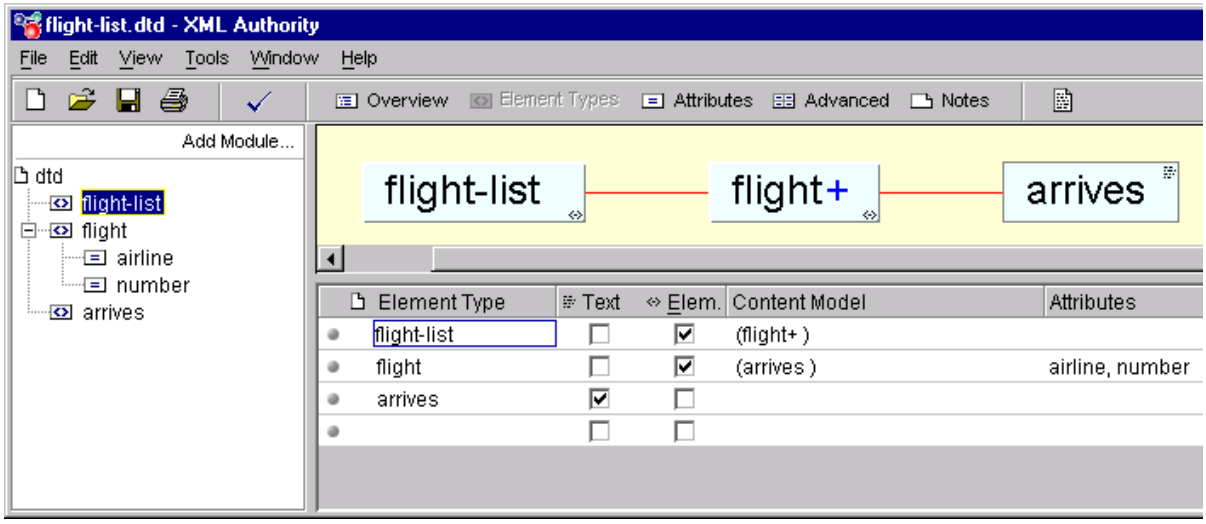
Transforming XML Datagrams into an Alternative XML Format

If the canonical <ROWSET> and <ROW> XML output from Figure 9–2 is not the XML format you need, then you can associate an XSLT stylesheet to your XSQL page template to transform this XML "datagram" in the server before returning the information in any alternative format desired.

When exchanging data with another program, typically you will agree in advance with the other party on a specific Document Type Descriptor (DTD) that describes the XML format you will be exchanging. A DTD is in effect, a "schema" definition. It formally defines what XML elements and attributes that a document of that type can have.

Let's assume you are given the `flight-list.dtd` definition and are told to produce your list of arriving flights in a format compliant with that DTD. You can use a visual tool such as Extensibility's "XML Authority" to browse the structure of the flight-list DTD as shown in Figure 9–3.

Figure 9–3 Exploring the "industry standard" flight-list.dtd using Extensibility's XML Authority



This shows that the standard XML formats for Flight Lists are:

- `<flight-list>` element, containing one or more…

- <flight> elements, having attributes airline and number, each of which contains an…
- <arrives> element.

By associating the following XSLT stylesheet, flight-list.xsl, with the XSQL page, you can change the default <ROWSET> and <ROW> format of your arriving flights into the "industry standard" DTD format.

```
<!-- XSLT Stylesheet to transform ROWSET/ROW results into flight-list format -->
<flight-list xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
             xsl:version="1.0">
  <xsl:for-each select="ROWSET/ROW">
      <flight airline="{CARRIER}" number="{FLIGHTNUMBER}">
        <arrives><xsl:value-of select="DUE"/></arrives>
      </flight>
  </xsl:for-each>
</flight-list>
```

The stylesheet is a template that includes the literal elements that you want produced in the resulting document, such as, <flight-list>, <flight>, and <arrives>, interspersed with special XSLT "actions" that allow you to do the following:

- Loop over matching elements in the source document using <xsl:for-each>
- Plug in the values of source document elements where necessary using <xsl:value-of>
- Plug in the values of source document elements into attribute values using {something}

Note two things have been added to the top-level <flight-list> element in the stylesheet:

- xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

  This defines the XML Namespace (xmlns) named "xsl" and identifies the uniform resource locator string that uniquely identifies the XSLT specification. Although it looks just like a URL, think of the string http://www.w3.org/1999/XSL/Transform as the "global primary key" for the set of elements that are defined in the XSLT 1.0 specification. Once the namespace is defined, we can then make use of the <xsl:XXX> action elements in our stylesheet to loop and plug values in where necessary.
- xsl:version="1.0"

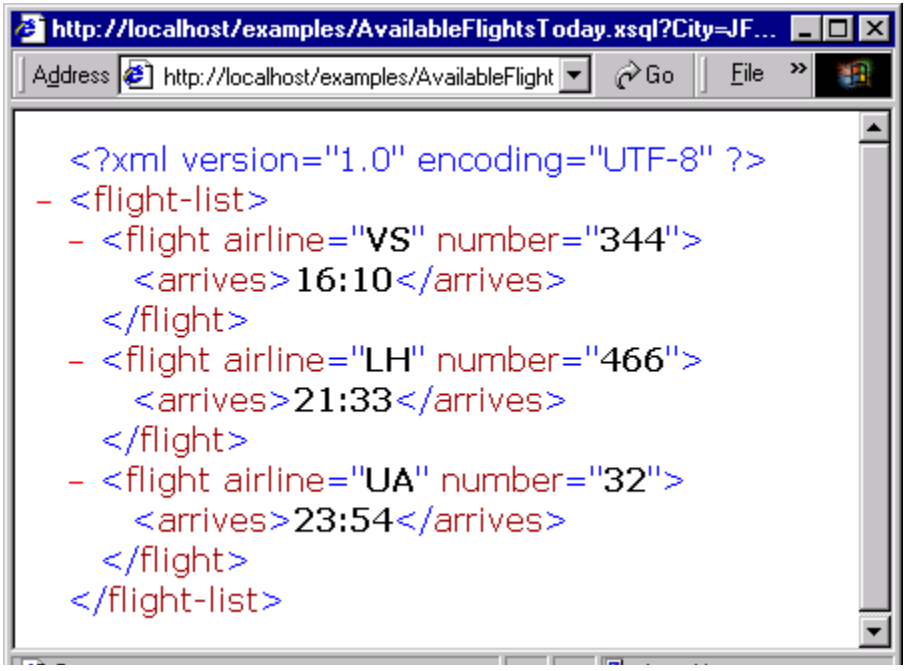
This attribute identifies the document as an XSLT 1.0 stylesheet. A version attribute is required on all XSLT Stylesheets for them to be valid and recognized by an XSLT Processor.

Associate the stylesheet to your XSQL Page by adding an `<?xml-stylesheet?>` processing instruction to the top of the page as follows:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="flight-list.xsl"?>
<xsql:query connection="demo" bind-params="City" xmlns:xsql="urn:oracle-xsql">
    SELECT Carrier, FlightNumber, Origin, TO_CHAR(ExpectedTime,'HH24:MI') AS Due
      FROM FlightSchedule
     WHERE TRUNC(ExpectedTime) = TRUNC(SYSDATE) AND Arrived = 'N'
       AND Destination = ?   /* The ? is a bind variable being bound */
      ORDER BY ExpectedTime  /* to the value of the City parameter   */
</xsql:query>
```

This is the W3C Standard mechanism of associating stylesheets with XML documents (http://www.w3.org/TR/xml-stylesheet). Specifying an associated XSLT stylesheet to the XSQL page causes the requesting program or browser to see the XML in the "industry-standard" format as specified by `flight-list.dtd` you were given as shown in Figure 9–4.

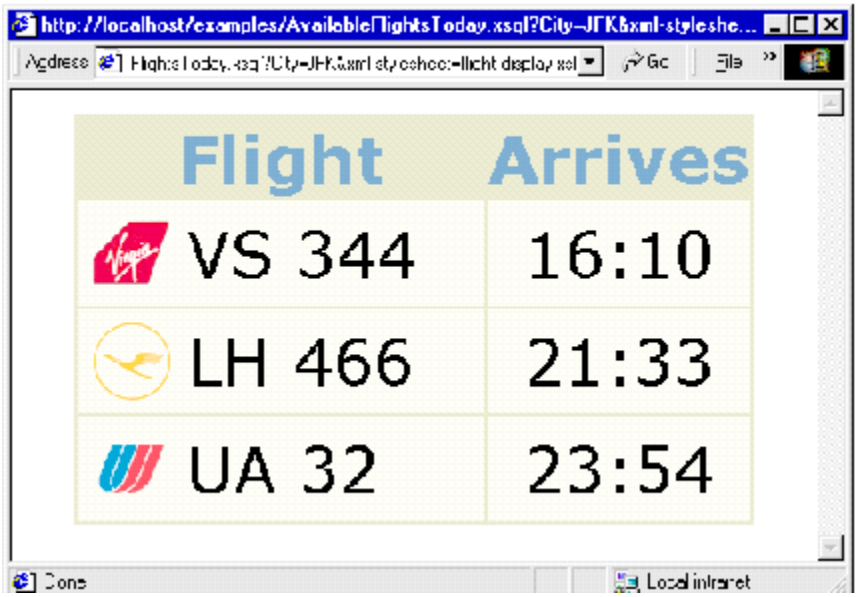
Figure 9–4 XSQL Page Results in "industry standard" XML Format



Transforming XML Datagrams into HTML for Display

To return the same XML information in HTML instead of an alternative XML format, simply use a different XSLT stylesheet. Rather than producing elements like `<flight-list>` and `<flight>`, your stylesheet produces HTML elements like `<table>`, `<tr>`, and `<td>` instead. The result of the dynamically queried information would then look like the HTML page shown in Figure 9–5. Instead of returning “raw” XML information, the XSQL Page leverages server-side XSLT transformation to format the information as HTML for delivery to the browser.

Figure 9–5 Using an Associated XSLT Stylesheet to Render HTML



Similar to the syntax of the `flight-list.xsl` stylesheet, the `flight-display.xsl` stylesheet looks like a template HTML page, with `<xsl:for-each>`, `<xsl:value-of>` and attribute value templates like `{DUE}` to plug in the dynamic values from the underlying `<ROWSET>` and `<ROW>` structured XML query results.

```
<!-- XSLT Stylesheet to transform ROWSET/ROW results into HTML -->
<html xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xsl:version="1.0">
  <head><link rel="stylesheet" type="text/css" href="flights.css" /></head>
  <body>
    <center><table border="0">
      <tr><th>Flight</th><th>Arrives</th></tr>
      <xsl:for-each select="ROWSET/ROW">
        <tr>
          <td>
            <table border="0" cellspacing="0" cellpadding="4">
              <tr>
                <td><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>
```

> **Note:** The stylesheet looks exactly like HTML, with one tiny difference. It is well-formed HTML. This means that each opening tag is properly closed (for example, <td>…</td>) and that empty tags use the XML empty element syntax
 instead of just
.

You can see that by combining the power of:

- Parameterized SQL statements to select any information you need from our Oracle database,
- Industry-standard XML as a portable, interim data exchange format
- XSLT to transform XML-based "data pages" into any XML- or HTML-based format you need

you can achieve very interesting and useful results quickly. You will see in later sections that what you have seen earlier is just scratching the surface of what you can do using XSQL pages.

Note: For a detailed introduction to XSLT and a thorough tutorial on how to apply XSLT to many different Oracle database scenarios, see "Building Oracle XML Applications", by Steve Muench, from O'Reilly and Associates.

Setting Up and Using XSQL Pages in Your Environment

You can develop and use XSQL pages in a variety of ways. We start by describing the easiest way to get started, using Oracle JDeveloper, then cover the details you'll need to understand to use XSQL pages in your production environment.

Using XSQL Pages with Oracle JDeveloper

The easiest way to work with XSQL pages during development is to use Oracle JDeveloper. Versions 3.1 and higher of the JDeveloper IDE support color-coded syntax highlighting, XML syntax checking, and easy testing of your XSQL pages. In addition, the JDeveloper 3.2 release supports debugging XSQL pages and adds new wizards to help create XSQL actions.

To create an XSQL page in a JDeveloper project, you can:

- Click the plus icon at the top of the navigator to add a new or existing XSQL page to your project
- Select *File* | *New...* and select "XSQL" from the "Web Objects" tab of the gallery

To get assistance adding XSQL action elements like `<xsql:query>` to your XSQL page, place the cursor where you want the new element to go and either:

- Select *XSQL Element...* from the right mouse menu, or
- Select *Wizards* | *XSQL Element...* from the IDE menu.

The XSQL Element wizard takes you through the steps of selecting which XSQL action you want to use, and which attributes you need to provide.

To syntax-check an XSQL page template, you can select *Check XML Syntax...* at any time from the right-mouse menu in the navigator after selecting the name of the XSQL page you'd like to check. If there are any XML syntax errors, they will appear in the message view and your cursor will be brought to the first one.

To test an XSQL page, simply select the page in the navigator and choose *Run* from the right-mouse menu. JDeveloper automatically starts up a local Web-to-go web server, properly configured to run XSQL pages, and tests your page by launching your default browser with the appropriate URL to request the page. Once you've run the XSQL page, you can continue to make modifications to it in the IDE — as well as to any XSLT stylesheets with which it might be associated — and after saving the files in the IDE you can immediately refresh the browser to observe the effect of the changes.

Using JDeveloper, the "XSQL Runtime" library should be added to your project's library list so that the CLASSPATH is properly setup. The IDE adds this entry automatically when you go through the New Object gallery to create a new XSQL page, but you can also add it manually to the project by selecting *Project | Project Properties...* and clicking on the "Libraries" tab.

Setting the CLASSPATH Correctly in Your Production Environment

Outside of the JDeveloper environment, you need to make sure that the XSQL page processor engine is properly configured to run. Oracle9*i* comes with the XSQL Servlet pre-installed to the Oracle HTTP Server that accompanies the database, but using XSQL in any other environment, you'll need to ensure that the Java CLASSPATH is setup correctly.

There are three "entry points" to the XSQL page processor:

- `oracle.xml.xsql.XSQLServlet`, the servlet interface
- `oracle.xml.xsql.XSQLCommandLine`, the command-line interface
- `oracle.xml.xsql.XSQLRequest`, the programmatic interface

Since all three of these interfaces, as well as the core XSQL engine itself, are written in Java, they are very portable and very simple to setup. The only setup requirements are to make sure the appropriate JAR files are in the CLASSPATH of the JavaVM that will be running processing the XSQL Pages. The JAR files include:

- `oraclexsql.jar`, the XSQL page processor
- `xmlparserv2.jar`, the Oracle XML Parser for Java v2
- `xsu12.jar`, the Oracle XML SQL utility
- `classes12.jar`, the Oracle JDBC driver

In addition, the *directory* where XSQL Page Processor's configuration file `XSQLConfig.xml` resides must also be listed as a directory in the CLASSPATH.

Putting all this together, if you have installed the XSQL distribution in `C:\xsql`, then your CLASSPATH would appear as follows:

```
C:\xsql\lib\classes12.classes12.jar;C:\xsql\lib\xmlparserv2.jar;
C:\xsql\lib\xsu12.jar;C:\xsql\lib\oraclexsql.jar;
directory_where_XSQLConfig.xml_resides
```

On Unix, if you extracted the XSQL distribution into your `/web` directory, the CLASSPATH would appear as follows:

```
/web/xsql/lib/classes12.jarclasses12.jar:/web/xsql/lib/xmlparserv2.jar:
/web/xsql/lib/xsu12.jar:/web/xsql/lib/oraclexsql.jar:
directory_where_XSQLConfig.xml_resides
```

To use the XSQL Servlet, one additional setup step is required. You must associate the `.xsql` file extension with the XSQL Servlet's java class `oracle.xml.xsql.XSQLServlet`. How you set the CLASSPATH of the web server's servlet environment and how you associate a Servlet with a file extension are done differently for each web server. The XSQL Servlet's Release Notes contain detailed setup information for specific web servers you might want to use with XSQL Pages.

Setting Up the Connection Definitions

XSQL pages refer to database connections by using a "nickname" for the connection defined in the XSQL configuration file. Connection names are defined in the `<connectiondefs>` section of `XSQLConfig.xml` file like this:

```
 <connectiondefs>
   <connection name="demo">
     <username>scott</username>
     <password>tiger</password>
     <dburl>jdbc:oracle:thin:@localhost:1521:testDB</dburl>
     <driver>oracle.jdbc.driver.OracleDriver</driver>
     <autocommit>true</autocommit>
   </connection>
   <connection name="lite">
     <username>system</username>
     <password>manager</password>
     <dburl>jdbc:Polite:POlite</dburl>
     <driver>oracle.lite.poljdbc.POLJDBCDriver</driver>
    </connection>
</connectiondefs>
```

For each connection, you can specify five pieces of information:

1. <username>
2. <password>
3. <dburl>, the JDBC connection string
4. <driver>, the fully-qualified class name of the JDBC driver to use
5. <autocommit>, optionally forces the autocommit to true or false

If the <autocommit> element is omitted, then the XSQL page processor will use the JDBC driver's default setting of the AutoCommit flag.

Any number of <connection> elements can be placed in this file to define the connections you need. An individual XSQL page refers to the connection it wants to use by putting a connection="*xxx*" attribute on the top-level element in the page (also called the "document element").

Note: For security reasons, when installing XSQL Servlet on your production web server, make sure the XSQLConfig.xml file does *not* reside in a directory that is part of the web server's virtual directory hierarchy. Failure to take this precaution risks exposing your configuration information over the web.

Using the XSQL Command-Line Utility

Often the content of a dynamic page will be based on data that is not frequently changing in your environment. To optimize performance of your web publishing, you can use operating system facilities to schedule offline processing of your XSQL pages, leaving the processed results to be served statically by your web server.

You can process any XSQL page from the command line using the XSQL command-line utility. The syntax is:

```
$ java oracle.xml.xsql.XSQLCommandLine xsqlpage [outfile] [param1=value1 ...]
```

If an *outfile* is specified, the result of processing *xsqlpage* is written to it, otherwise the result goes to standard out. Any number of parameters can be passed to the XSQL page processor and are available for reference by the XSQL page being processed as part of the request. However, the following parameter names are recognized by the command-line utility and have a pre-defined behavior:

- xml-stylesheet=*stylesheetURL*

Provides the relative or absolute URL for a stylesheet to use for the request. Also can be set to the string `none` to suppress XSLT stylesheet processing for debugging purposes.

- `posted-xml=`*`XMLDocumentURL`*

  Provides the relative or absolute URL of an XML resource to treat as if it were posted as part of the request.

- `useragent=`*`UserAgentString`*

  Used to simulate a particular HTTP User-Agent string from the command line so that an appropriate stylesheet for that User-Agent type will be selected as part of command-line processing of the page.

The ?/xdk/java/xsql/bin directory contains a platform-specific command script to automate invoking the XSQL command-line utility. This script sets up the Java runtime to run `oracle.xml.xsql.XSQLCommandLine` class.

Overview of All XSQL Pages Capabilities

So far we've only seen a single XSQL action element, the `<xsql:query>` action. This is by far the most popular action, but it is not the only one that comes built-in to the XSQL Pages framework. We explore the full set of functionality that you can exploit in your XSQL pages in the following sections.

Using All of the Core Built-in Actions

This section provides a list of the core built-in actions, including a brief description of what each action does, and a listing of all required and optional attributes that each supports.

The <xsql:query> Action

The `<xsql:query>` action element executes a SQL select statement and includes a canonical XML representation of the query's result set in the data page. This action requires a database connection to be provided by supplying a `connection="`*`connname`*`"` attribute on the document element of the XSQL page in which it appears.

The syntax for the action is:

```
<xsql:query>
   SELECT Statement
</xsql:query>
```

Any legal SQL select statement is allowed. If the select statement produces no rows, a "fallback" query can be provided by including a nested <xsql:no-rows-query> element like this:

```
<xsql:query>
  SELECT Statement
  <xsql:no-rows-query>
    SELECT Statement to use if outer query returns no rows
  </xsql:no-rows-query>
</xsql:query>
```

An <xsql:no-rows-query> element can *itself* contain nested <xsql:no-rows-query> elements to any level of nesting. The options available on the <xsql:no-rows-query> are identical to those available on the <xsql:query> action element.

By default, the XML produced by a query will reflect the column structure of its resultset, with element names matching the names of the columns. Columns in the result with nested structure like:

- Object Types
- Collection Types
- CURSOR Expressions

produce nested elements that reflect this structure. The result of a typical query containing different types of columns and returning one row might look like this:

```
<ROWSET>
  <ROW id="1">
    <VARCHARCOL>Value</VARCHARCOL>
    <NUMBERCOL>12345</NUMBERCOL>
    <DATECOL>12/10/2001 10:13:22</DATECOL>
    <OBJECTCOL>
       <ATTR1>Value</ATTR1>
       <ATTR2>Value</ATTR2>
    </OBJECTCOL>
    <COLLECTIONCOL>
       <COLLECTIONCOL_ITEM>
         <ATTR1>Value</ATTR1>
         <ATTR2>Value</ATTR2>
       </COLLECTIONCOL_ITEM>
       <COLLECTIONCOL_ITEM>
         <ATTR1>Value</ATTR1>
         <ATTR2>Value</ATTR2>
       </COLLECTIONCOL_ITEM>
```

```
    </COLLECTIONCOL>
    <CURSORCOL>
      <CURSORCOL_ROW>
        <COL1>Value1</COL1>
        <COL2>Value2</COL2>
      </CURSORCOR_ROW>
    </CURSORCOL>
  </ROW>
</ROWSET>
```

A <ROW> element will repeat for each row in the result set. Your query can use standard SQL column aliasing to rename the columns in the result, and in doing so effectively rename the XML elements that are produced as well. Note that such column aliasing is *required* for columns whose names would otherwise be an illegal name for an XML element.

For example, an <xsql:query> action like this:

```
<xsql:query>SELECT TO_CHAR(hiredate,'DD-MON') FROM EMP</xsql:query>
```

would produce an error because the default column name for the calculated expression will be an illegal XML element name. You can fix the problem with column aliasing like this:

```
<xsql:query>SELECT TO_CHAR(hiredate,'DD-MON') as hiredate FROM EMP</xsql:query>
```

The optional attributes listed in Table 9–1 can be supplied to control various aspects of the data retrieved and the XML produced by the <xsql:query> action.

Table 9–1 Attributes for <xsql:query>

Attribute Name	Description
bind-params = "*string*"	Ordered, space-delimited list of one or more XSQL parameter names whose values will be used to bind to the JDBC bind variable in the appropriate sequential position in the SQL statement.
date-format = "*string*"	Date format mask to use for formatted date column / attribute values in XML being queried. Valid values are those documented for the `java.text.SimpleDateFormat` class.
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` and `no`. The default value is `yes`.

Table 9–1 Attributes for <xsql:query>

Attribute Name	Description
fetch-size = "*integer*"	Number of records to fetch in each round-trip to the database. If not set, the default value is used as specified by the `/XSQLConfig/processor/default-fetch-size` configuration setting in `XSQLConfig.xml`
id-attribute = "*string*"	XML attribute name to use instead of the default `num` attribute for uniquely identifying each row in the result set. If the value of this attribute is the empty string, the row id attribute is suppressed.
id-attribute-column = "*string*"	Case-sensitive name of the column in the result set whose value should be used in each row as the value of the row id attribute. The default is to use the row count as the value of the row id attribute.
include-schema = "*boolean*"	If set to `yes`, includes an inline XML schema that describes the structure of the result set. Valid values are `yes` and `no`. The default value is `no`.
max-rows = "*integer*"	Maximum number of rows to fetch, after optionally skipping the number of rows indicated by the `skip-rows` attribute. If not specified, default is to fetch all rows.
null-indicator = "*boolean*"	Indicates whether to signal that a column's value is NULL by including the `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`. The default value is `no`.
row-element = "*string*"	XML element name to use instead of the default `<ROW>` element name for the entire 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>` element name for the entire 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.

The <xsql:dml> Action

You can use the <xsql:dml> action to perform any DML or DDL operation, as well as any PL/SQL block. This action requires a database connection to be provided by supplying a connection="*connname*" attribute on the document element of the XSQL page in which it appears.
The syntax for the action is:

```
<xsql:dml>
  DML Statement or DDL Statement or PL/SQL Block
</xsql:dml>
```

Table 9–2 lists the optional attributes that you can use on the <xsql:dml> action.

Table 9–2 Attributes for <xsql:dml>

Attribute Name	Description
commit = "*boolean*"	If set to yes, calls commit on the current connection after a successful execution of the DML statement. Valid values are yes and no. The default value is no.
bind-params = "*string*"	Ordered, space-delimited list of one or more XSQL parameter names whose values will be used to bind to the JDBC bind variable in the appropriate sequential position in the SQL statement.
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 and no. The default value is yes.

The <xsql:ref-cursor-function> Action

The <xsql:ref-cursor-function> action allows you to include the XML results produced by a query whose result set is determined by executing a PL/SQL stored function. This action requires a database connection to be provided by supplying a connection="*connname*" attribute on the document element of the XSQL page in which it appears.

By exploiting PL/SQL's dynamic SQL capabilities, the query can be dynamically and/or conditionally constructed by the function before a cursor handle to its result set is returned to the XSQL page processor. As its name implies, the return value of the function being invoked must be of type REF CURSOR.

The syntax of the action is:

```
<xsql:ref-cursor-function>
  [SCHEMA.][PACKAGE.]FUNCTION_NAME(args);
</xsql:ref-cursor-function>
```

With the exception of the `fetch-size` attribute, the optional attributes available for the `<xsql:ref-cursor-function>` action are exactly the same as for the `<xsql:query>` action that are listed Table 9–1.

For example, consider the 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
   -- Conditionally return a dynamic query as a REF CURSOR
   IF id = 1 THEN
     OPEN the_cursor
      FOR 'SELECT empno, ename FROM EMP'; -- An EMP Query
   ELSE
     OPEN the_cursor
      FOR 'SELECT dname, deptno FROM DEPT'; -- A DEPT Query
   END IF;
   RETURN the_cursor;
  END;
END;
```

An `<xsql:ref-cursor-function>` can include the dynamic results of the REF CURSOR returned by this function by doing:

```
<xsql:ref-cursor-function>
  DynCursor.DynamicQuery(1);
</xsql:ref-cursor-function>
```

The <xsql:include-owa> Action

The `<xsql:include-owa>` action allows you to include XML content that has been generated by a database stored procedure. This action requires a database connection to be provided by supplying a connection="*connname*" attribute on the document element of the XSQL page in which it appears.

The stored procedure uses the standard Oracle Web Agent (OWA) packages (HTP and HTF) to "print" the XML tags into the server-side page buffer, then the XSQL page processor fetches, parses, and includes the dynamically-produced XML content in the data page. The stored procedure must generate a well-formed XML page or an appropriate error is displayed.

The syntax for the action is:

```
<xsql:include-owa>
   PL/SQL Block invoking a procedure that uses the HTP and/or HTF packages
</xsql:include-owa>
```

Table 9–3 lists the optional attributes supported by this action.

Table 9–3 Attributes for <xsql:include-owa>

Attribute Name	Description
bind-params = "*string*"	Ordered, space-delimited list of one or more XSQL parameter names whose values will be used to bind to the JDBC bind variable in the appropriate sequential position in the SQL statement.
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 and no. The default value is yes.

Using Bind Variables

To parameterize the results of any of the preceding actions, you can use SQL bind variables. This allows your XSQL page template to produce different results based on the value of parameters passed in the request. To use a bind variable, simply include a question mark anywhere in the statement where bind variables are allowed by SQL. For example, your <xsql:query> action might contain the select statement:

```
SELECT s.ticker as "Symbol", s.last_traded_price as "Price"
  FROM latest_stocks s, customer_portfolio p
 WHERE p.customer_id = ?
   AND s.ticker = p.ticker
```

Using a question mark to create a bind-variable for the customer id. Whenever the SQL statement is executed in the page, parameter values are bound to the bind variable by specifying the bind-params attribute on the action element. Using the example preceding, we could create an XSQL page that binds the indicated bind variables to the value of the custid parameter in the page request like this:

```
<!-- CustomerPortfolio.xsql -->
<portfolio 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>
```

The XML data for a particular customer's portfolio can then be requested by passing the customer id parameter in the request like this:

```
http://yourserver.com/fin/CustomerPortfolio.xsql?custid=1001
```

The value of the `bind-params` attribute is a space-delimited list of parameter names whose left-to-right order indicates the positional bind variable to which its value will be bound in the statement. So, if your SQL statement has five question marks, then your bind-params attribute needs a space-delimited list of five parameter names. If the same parameter value needs to be bound to several different occurrences of a question-mark-indicated bind variable, you simply repeat the name of the parameters in the value of the `bind-params` attribute at the appropriate position. Failure to include *exactly* as many parameter names in the bind-params attribute as there are question marks in the query, will results in an error when the page is executed.

Bind variables can be used in any action that expects a SQL statement. The following page gives additional examples:

```
<!-- CustomerPortfolio.xsql -->
<portfolio 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>
  </analysis>
</portfolio>
```

Using Lexical Substitution Parameters

For any XSQL action element, you can substitute the value of any attribute, or the text of any contained SQL statement, by using a lexical substitution parameter. This allows you to parameterize how the actions behave as well as substitute parts of the SQL statements they perform. Lexical substitution parameters are referenced using the syntax {*@ParameterName*}.

The following example illustrates using two *lexical* substitution parameters, one which allows the maximum number of rows to be passed in as a parameter, and the other which controls the list of columns to ORDER BY.

```
<!-- DevOpenBugs.xsql -->
<open-bugs connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:query max-rows="{@max}" bind-params="dev prod">
    SELECT bugno, abstract, status
      FROM bug_table
     WHERE programmer_assigned = UPPER(?)
       AND product_id          = ?
       AND status < 80
    ORDER BY {@orderby}
  </xsql:query>
</open-bugs>
```

This example could then show the XML for a given developer's open bug list by requesting the URL:

```
http://yourserver.com/bug/DevOpenBugs.xsql?dev=smuench&prod=817
```

or using the XSQL Command-Line Utility to request:

```
$ xsql DevOpenBugs.xsql dev=smuench prod=817
```

We close by noting that lexical parameters can also be used to parameterize the XSQL page connection, as well as parameterize the stylesheet that is used to process the page like this:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="{@sheet}.xsl"?>
<!-- DevOpenBugs.xsql -->
<open-bugs connection="{@conn}" xmlns:xsql="urn:oracle-xsql">
  <xsql:query max-rows="{@max}" bind-params="dev prod">
```

```
    SELECT bugno, abstract, status
      FROM bug_table
     WHERE programmer_assigned = UPPER(?)
       AND product_id          = ?
       AND status < 80
    ORDER BY {@orderby}
  </xsql:query>
</open-bugs>
```

Providing Default Values for Bind Variables and Parameters

It is often convenient to provide a default value for a bind variable or a substitution parameter directly in the page. This allows the page to be parameterized without requiring the requester to explicitly pass in all the values in each request.

To include a default value for a parameter, simply add an XML attribute of the same name as the parameter to the action element, or to any ancestor element. If a value for a given parameter is not included in the request, the XSQL page processor looks for an attribute by the same name on the current action element. If it doesn't find one, it keeps looking for such an attribute on each ancestor element of the current action element until it gets to the document element of the page.

As a simple example, the following page defaults the value of the `max` parameter to 10 for both `<xsql:query>` actions in the page:

```
<example max="10" connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:query max-rows="{@max}">SELECT * FROM TABLE1</xsql:query>
  <xsql:query max-rows="{@max}">SELECT * FROM TABLE2</xsql:query>
</example>
```

This example defaults the first query to have a `max` of 5, the second query to have a `max` of 7 and the third query to have a `max` of 10.

```
<example max="10" connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:query max="5" max-rows="{@max}">SELECT * FROM TABLE1</xsql:query>
  <xsql:query max="7" max-rows="{@max}">SELECT * FROM TABLE2</xsql:query>
  <xsql:query max-rows="{@max}">SELECT * FROM TABLE3</xsql:query>
</example>
```

Of course, all of these defaults would be overridden if a value of max is supplied in the request like:

```
http://yourserver.com/example.xsql?max=3
```

Bind variables respect the same defaulting rules so a — not-very-useful, yet educational — page like this:

```
<example val="10" connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:query tag-case="lower" bind-params="val val val">
    SELECT ? as somevalue
      FROM DUAL
     WHERE ? = ?
  </xsql:query>
</example>
```

Would return the XML datagram:

```
<example>
  <rowset>
    <row>
      <somevalue>10</somevalue>
    </row>
  </row>
</example>
```

if the page were requested without any parameters, while a request like:

`http://yourserver.com/example.xsql?val=3`

Would return:

```
<example>
  <rowset>
    <row>
      <somevalue>3</somevalue>
    </row>
  </row>
</example>
```

To illustrate an important point for bind variables, imagine removing the default value for the `val` parameter from the page by removing the `val` attribute like this:

```
<example connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:query tag-case="lower" bind-params="val val val">
    SELECT ? as somevalue
      FROM DUAL
     WHERE ? = ?
  </xsql:query>
</example>
```

Now a request for the page without supplying any parameters would return:

```
<example>
  <rowset/>
</example>
```

because a bind variable that is bound to a parameter with *neither* a default value *nor* a value supplied in the request will be bound to NULL, causing the WHERE clause in our example page preceding to return no rows.

Understanding the Different Kinds of Parameters

XSQL pages can make use of parameters supplied in the request, as well as page-private parameters whose names and values are determined by actions in the page. If an action encounters a reference to a parameter named *param* in either a `bind-params` attribute or in a lexical parameter reference, the value of the *param* parameter is resolved by using:

1. The value of the page-private parameter named *param*, if set, otherwise
2. The value of the request parameter named *param*, if supplied, otherwise
3. The default value provided by an attribute named *param* on the current action element or one of its ancestor elements, otherwise
4. The value NULL for bind variables and the empty string for lexical parameters

For XSQL pages that are processed by the XSQL Servlet over HTTP, two additional HTTP-specific type of parameters are available to be set and referenced. These are HTTP-Session-level variables and HTTP Cookies. For XSQL pages processed through the XSQL Servlet, the parameter value resolution scheme is augmented as follows. The value of a parameter *param* is resolved by using:

1. The value of the page-private parameter *param*, if set, otherwise
2. The value of the cookie named *param*, if set, otherwise
3. The value of the session variable named *param*, if set, otherwise
4. The value of the request parameter named *param*, if supplied, otherwise
5. The default value provided by an attribute named *param* on the current action element or one of its ancestor elements, otherwise
6. The value NULL for bind variables and the empty string for lexical parameters

The resolution order is arranged this way so that users cannot supply parameter values in a request to override parameters of the same name that have been set in

the HTTP session — whose lifetime is the duration of the HTTP session and controlled by your web server — or set as cookies, which can bet set to "live" across browser sessions.

The <xsql:include-request-params> Action

The `<xsql:include-request-params>` action allows you to include an XML representation of all parameters in the request in your datagram. This is useful if your associated XSLT stylesheet wants to refer to any of the request parameter values by using XPath expressions.

The syntax of the action is:

```
<xsql:include-request-params/>
```

The XML included will have the form:

```
<request>
  <parameters>
    <paramname>value1</paramname>
    <ParamName2>value2</ParamName2>
      :
  </parameters>
</request>
```

or the form:

```
<request>
  <parameters>
    <paramname>value1</paramname>
    <ParamName2>value2</ParamName2>
      :
  </parameters>
  <session>
    <sessVarName>value1</sessVarName>
      :
  </session>
  <cookies>
    <cookieName>value1</cookieName>
      :
  </cookies>
</request>
```

when processing pages through the XSQL Servlet.

This action has no required or optional attributes.

The <xsql:include-param> Action

The `<xsql:include-param>` action allows you to include an XML representation of a single parameter in your datagram. This is useful if your associated XSLT stylesheet wants to refer to the parameter's value by using an XPath expression.

The syntax of the action is:

```
<xsql:include-param name="paramname" />
```

This `name` attribute is required, and supplies the name of the parameter whose value you would like to include. This action has no optional attributes.

The XML included will have the form:

```
<paramname>value1</paramname>
```

The <xsql:include-xml> Action

The `<xsql:include-xml>` action includes the XML contents of a local, remote, or database-driven XML resource into your datagram. The resource is specified either by URL or a SQL statement.

The syntax for this action is:

```
<xsql:include-xml href="URL"/>
```

or

```
<xsql:include-xml>
  SQL select statement selecting a single row containing a single
  CLOB or VARCHAR2 column value
</xsql:include-xml>
```

The URL can be an absolute, http-based URL to retrieve XML from another web site, or a relative URL. The `href` attribute and the SQL statement are mutually exclusive. If one is provided the other is not allowed.

Table 9–5 lists the attributes supported by this action. Attributes in bold are required.

Table 9–4 Attributes for <xsql:include-xml>

Attribute Name	Description
bind-params = "*string*"	Ordered, space-delimited list of one or more XSQL parameter names whose values will be used to bind to the JDBC bind variable in the appropriate sequential position in the SQL statement.

The <xsql:include-posted-xml> Action

The `<xsql:include-posted-xml>` action includes the XML document that has been posted in the request into the XSQL page. If an HTML form is posted instead of an XML document, the XML included will be similar to that included by the `<xsql:include-request-params>` action.

The <xsql:set-page-param> Action

The `<xsql:set-page-param>` action sets a page-private parameter to a value. The value can be supplied by a combination of static text and other parameter values, or alternatively from the result of a SQL select statement.

The syntax for this action is:

```
<xsql:set-page-param name="paramname" value="value"/>
```

or

```
<xsql:set-page-param name="paramname">
  SQL select statement
</xsql:set-page-param>
```

or

```
<xsql:set-page-param name="paramname" xpath="XPathExpression"/>
```

If you use the SQL statement option, a single row is fetched from the result set and the parameter is assigned the value of the *first* column. This 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 Page 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.

Once a page-private parameter is set, subsequent action handlers can use this value as a lexical parameter, for example `{@po_id}`, or as a SQL bind parameter value by referencing its name in the `bind-params` attribute of any action handler that supports SQL operations.

If you need to set *several* session parameter values based on the results of a single SQL statement, instead of using the `name` attribute, you can use the `names` attribute

and supply a space-or-comma-delimited list of one or more session parameter names. For example:

```
<xsql:set-page-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-page-param>
```

Either the `name` or the `names` attribute is required. The `value` attribute and the contained SQL statement are mutually exclusive. If one is supplied, the other must not be.

Table 9–5 lists the attributes supported by this action. Attributes in bold are required.

Table 9–5 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 whose values will be used to bind to the JDBC bind variable in the appropriate sequential position in the SQL statement.
ignore-empty-value = "*boolean*"	Indicates whether the page-level parameter assignment should be ignored if the value to which it is being assigned is an empty string. Valid values are `yes` and `no`. The default value is `no`.
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 Page Processor.

The <xsql:set-session-param> Action

The `<xsql:set-session-param>` action sets an HTTP session-level parameter to a value. The value of the session-level parameter remains for the lifetime of the current browser user's HTTP session, which is controlled by the web server. The value can be supplied by a combination of static text and other parameter values, or alternatively from the result of a SQL select statement.

Since this feature is specific to Java Servlets, this action is only effective if the XSQL page in which it appears is being processed by the XSQL Servlet. If this action is encountered in an XSQL page being processed by the XSQL command-line utility or the `XSQLRequest` programmatic API, this action is a no-op.

The syntax for this action is:

```
<xsql:set-session-param name="paramname" value="value"/>
```

or

```
<xsql:set-session-param name="paramname">
  SQL select statement
</xsql:set-session-param>
```

If you use the SQL statement option, a single row is fetched from the result set and the parameter is assigned the value of the *first* column. This use requires a database connection to be provided by supplying a `connection="connname"` attribute on the document element of the XSQL page in which it appears.

If you need to set several session parameter values based on the results of a single SQL statement, instead of using the `name` attribute, you can use the `names` attribute and supply a space-or-comma-delimited list of one or more session parameter names. For example:

```
<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>
```

Either the `name` or the `names` attribute is required. The `value` attribute and the contained SQL statement are mutually exclusive. If one is supplied, the other must not be.

Table 9–6 lists the optional attributes supported by this action.

Table 9–6 Attributes for <xsql:set-session-param>

Attribute Name	Description
name = "*string*"	Name of the session-level variable whose value you want to set.
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.

Table 9–6 Attributes for <xsql:set-session-param>

Attribute Name	Description
bind-params = "*string*"	Ordered, space-delimited list of one or more XSQL parameter names whose values will be used to bind to the JDBC bind variable in the appropriate sequential position in the SQL statement.
ignore-empty-value = "*boolean*"	Indicates whether the session-level parameter assignment should be ignored if the value to which it is being assigned is an empty string. Valid values are `yes` and `no`. The default value is `no`.
only-if-unset = "*boolean*"	Indicates whether the session variable assignment should only occur when the session variable currently does not exists. Valid values are `yes` and `no`. The default value is `no`.

The <xsql:set-cookie> Action

The `<xsql:set-cookie>` action sets an HTTP cookie to a value. By default, the value of the cookie remains for the lifetime of the current browser, but its lifetime can be changed by supplying the optional `max-age` attribute. The value to be assigned to the cookie can be supplied by a combination of static text and other parameter values, or alternatively from the result of a SQL select statement.

Since this feature is specific to the HTTP protocol, this action is only effective if the XSQL page in which it appears is being processed by the XSQL Servlet. If this action is encountered in an XSQL page being processed by the XSQL command-line utility or the `XSQLRequest` programmatic API, this action is a no-op.

The syntax for this action is:

```
<xsql:set-cookie name="paramname" value="value"/>
```

or

```
<xsql:set-cookie name="paramname">
  SQL select statement
</xsql:set-cookie>
```

If you use the SQL statement option, a single row is fetched from the result set and the parameter is assigned the value of the *first* column. This use requires a database connection to be provided by supplying a `connection="connname"` attribute on the document element of the XSQL page in which it appears.

If you need to set several cookie values based on the results of a single SQL statement, instead of using the `name` attribute, you can use the `names` attribute and supply a space-or-comma-delimited list of one or more cookie names. For example:

```
<xsql:set-cookie names="paramname1 paramname2 paramname3">
  SELECT expression_or_column1, expression_or_column2, expression_or_column3
    FROM table
   WHERE clause_identifying_a_single_row
</xsql:set-cookie>
```

Either the `name` or the `names` attribute is required. The `value` attribute and the contained SQL statement are mutually exclusive. If one is supplied, the other must not be. The number of columns in the select list must match the number of cookies being set or an error message will result.

Table 9–7 lists the optional attributes supported by this action.

Table 9–7 Attributes for <xsql:set-cookie>

Attribute Name	Description
name = "*string*"	Name of the cookie whose value you want to set.
names = "*string string* ..."	Space-or-comma-delimited list of the cookie 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 whose values will be 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, then it defaults to the fully-qualified hostname (for example, `bigserver.yourcompany.com`) of the document creating the cookie.
ignore-empty-value = "*boolean*"	Indicates whether the cookie assignment should be ignored if the value to which it is being assigned is an empty string. Valid values are `yes` and `no`. The default value is `no`.
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 should only occur when the cookie currently does not exists. Valid values are `yes` and `no`. The default value is `no`.

Table 9–7 Attributes for <xsql:set-cookie>

Attribute Name	Description
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.
immediate = "*boolean*"	Indicates whether the cookie assignment should be 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. The default value is no.

The <xsql:set-stylesheet-param> Action

The `<xsql:set-stylesheet-param>` action sets a top-level XSLT stylesheet parameter to a value. The value can be supplied by a combination of static text and other parameter values, or alternatively from the result of a SQL select statement. The stylesheet parameter will be set on any stylesheet used during the processing of the current page.

The syntax for this action is:

```
<xsql:set-stylesheet-param name="paramname" value="value"/>
```

or

```
<xsql:set-stylesheet-param name="paramname">
  SQL select statement
</xsql:set-stylesheet-param>
```

If you use the SQL statement option, a single row is fetched from the result set and the parameter is assigned the value of the *first* column. This use requires a database connection to be provided by supplying a connection="*connname*" attribute on the document element of the XSQL page in which it appears.

If you need to set several stylesheet parameter values based on the results of a single SQL statement, instead of using the name attribute, you can use the names attribute and supply a space-or-comma-delimited list of one or more cookie names. For example:

```
<xsql:set-stylesheet-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-stylesheet-param>
```

Either the `name` or the `names` attribute is required. The `value` attribute and the contained SQL statement are mutually exclusive. If one is supplied, the other must not be.

Table 9–8 lists the optional attributes supported by this action.

Table 9–8 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. 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 whose values will be used to bind to the JDBC bind variable in the appropriate sequential position in the SQL statement.
ignore-empty-value = "*boolean*"	Indicates whether the stylesheet parameter assignment should be ignored if the value to which it is being assigned is an empty string. Valid values are `yes` and `no`. The default value is `no`.

Aggregating Information Using <xsql:include-xsql>

The `<xsql:include-xsql>` action makes it very easy to include the results of one XSQL page into another page. This allows you to easily aggregate content from a page that you've already built and repurpose it. The examples that follow illustrate two of the most common uses of `<xsql:include-xsql>`.

Assume you have an XSQL page that lists discussion forum categories:

```
<!-- Categories.xsql -->
<xsql:query connection="forum" xmlns:xsql="urn:oracle-xsql">
  SELECT name
    FROM categories
    ORDER BY name
</xsql:query>
```

You can include the results of this page into a page that lists the ten most recent topics in the current forum like this:

```
<!-- TopTenTopics.xsql -->
```

```
<top-ten-topics connection="forum" xmlns:xsql="urn:oracle-xsql">
  <topics>
    <xsql:query max-rows="10">
      SELECT subject FROM topics ORDER BY last_modified DESC
    </xsql:query>
  </topics>
  <categories>
    <xsql:include-xsql href="Categories.xsql"/>
  </categories>
</top-ten-topics>
```

You can use `<xsql:include-xsql>` to include an existing page to apply an XSLT stylesheet to it as well. So, if we have two different XSLT stylesheets:

- `cats-as-html.xsl`, which renders the topics in HTML, and
- `cats-as-wml.xsl`, which renders the topics in WML

Then one approach for catering to two different types of devices is to create different XSQL pages for each device. We can create:

```
<?xml version="1.0"?>
<!-- HTMLCategories.xsql -->
<?xml-stylesheet type="text/xsl" href="cats-as-html.xsl"?>
<xsql:include-xsql href="Categories.xsql" xmlns:xsql="urn:oracle-xsql"/>
```

which aggregates `Categories.xsql` and applies the `cats-as-html.xsl` stylesheet, and another page:

```
<?xml version="1.0"?>
<!-- WMLCategories.xsql -->
<?xml-stylesheet type="text/xsl" href="cats-as-html.xsl"?>
<xsql:include-xsql href="Categories.xsql" xmlns:xsql="urn:oracle-xsql"/>
```

which aggregates `Categories.xsql` and applies the `cats-as-wml.xsl` stylesheet for delivering to wireless devices. In this way, we've repurposed the reusable Categories.xsql page content in two different ways.

If the page being aggregated contains an `<?xml-stylesheet?>` processing instruction, then that stylesheet is applied before the result is aggregated, so using `<xsql:include-xsql>` you can also easily chain the application of XSLT stylesheets together.

When one XSQL page aggregates another page's content using `<xsql:include-xsql>` all of the request-level parameters are visible to the "nested" page. For pages processed by the XSQL Servlet, this also includes

session-level parameters and cookies, too. As you would expect, none of the aggregating page's *page-private* parameters are visible to the nested page.

Table 9–9 lists the attributes supported by this action. Required attributes are in bold.

Table 9–9 Attributes for <xsql:include-xsql>

Attribute Name	Description
href = "*string*"	Relative or absolute URL of XSQL page to be included.
reparse = "*boolean*"	Indicates whether output of included XSQL page should be *reparsed* before it is included. Useful if included XSQL page is selecting the *text* of an XML document fragment that the including page wants to treat as elements. Valid values are `yes` and `no`. The default value is `no`.

Including XMLType Query Results

Oracle9*i* introduces the XMLType for use with storing and querying XML-based database content. You can exploit database XML features to produce XML for inclusion in your XSQL pages using one of two techniques:

- `<xsql:query>` handles any query including columns of type XMLType, however it handles XML markup in CLOB/VARCHAR2 columns as literal text.
- `<xsql:include-xml>` *parses* and includes a single CLOB or String-based XML document retrieved from a query

The difference between the two approaches lies in the fact that the `<xsql:include-xml>` action parses the literal XML appearing in a CLOB or String-value to turn it on the fly into a tree of elements and attributes. On the other hand, using the `<xsql:query>` action, XML markup appearing in CLOB or String valued-columns is left as literal text.

Another difference is that while `<xsql:query>` can handle query results of any number of columns and rows, the `<xsql:include-xml>` is designed to work on a single column of a single row. Accordingly, when using `<xsql:include-xml>`, the SELECT statement that appears inside it should return 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 XML document will be parsed and included into your XSQL page.

The following example uses nested `xmlagg()` functions to aggregate the results of a dynamically-constructed XML document containing departments and nested

employees into a *single* XML "result" document, wrapped in a `<DepartmentList>` element:

```
<xsql:query connection="orcl92" xmlns:xsql="urn:oracle-xsql">
  select XmlElement("DepartmentList",
           XmlAgg(
             XmlElement("Department",
               XmlAttributes(deptno as "Id"),
               XmlForest(dname as "Name"),
               (select XmlElement("Employees",
                         XmlAgg(
                           XmlElement("Employee",
                             XmlAttributes(empno as "Id"),
                             XmlForest(ename as "Name",
                                       sal   as "Salary",
                                       job   as "Job")
                           )
                         )
                       )
                 from emp e
                where e.deptno = d.deptno
               )
             )
           )
         ) as result
   from dept d
  order by dname
</xsql:query>
```

Considering another example, suppose you have a number of <Movie> XML documents stored in a table of `XmlType` called `MOVIES`. Each document might look something like this:

```
<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>Gwenyth</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>
```

You can use the built-in Oracle9*i* XPath query features to extract an aggregate list of all cast members who have received Oscar awards from any movie in the database using a query like this:

```
select xmlelement("AwardedActors",
         xmlagg(extract(value(m),
                '/Movie/Cast/*[Award[@From="Oscar"]]')))
  from movies m
```

To include this query result of `XMLType` into your XSQL page, simply paste the query inside an `<xsql:query>` element, and make sure you include an alias for the query expression (for example "as result" following):

```
<xsql:query connection="orcl92" xmlns:xsql="urn:oracle-xsql">
  select xmlelement("AwardedActors",
           xmlagg(extract(value(m),
                  '/Movie/Cast/*[Award[@From="Oscar"]]'))) as result
    from movies m
</xsql:query>
```

Note that again we use the combination of `xmlelement()` and `xmlagg()` to have the database aggregate all of the XML fragments identified by the query into a single, well-formed XML document. `The combination of xmlelement() and xmlagg() work together to produce a well-formed result like this:`

```
<AwardedActors>
  <Actor>...</Actor>
  <Actress>...</Actress>
</AwardedActors>
```

Notice that you can use the standard XSQL Pages bind variable capabilities in the middle of an XPath expression, too, if you concatenate the bind variable into the expression. For example, to parameterize the value "Oscar" into a parameter named award-from, you could use an XSQL Page like this:

```
<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>
```

Handling Posted Information

In addition to simplifying the assembly and transformation of XML content, the XSQL Pages framework makes it easy to handle posted XML content as well. Built-in actions simplify the handling of posted information from both XML document and HTML forms, and allow that information to be posted directly into a database table using the underlying facilities of the Oracle XML SQL Utility.

The XML SQL Utility provides the ability to data database inserts, updates, and deletes based on the content of an XML document in "canonical" form with respect to a target table or view. For a given database table, the "canonical" XML form of its data is given by one row of XML output from a `SELECT * FROM tablename` query against it. Given an XML document in this canonical form, the XML SQL Utility can automate the insert, update, and / or delete for you. By combining the XML SQL Utility with an XSLT transformation, you can transform XML in any format into the canonical format expected by a given table, and then ask the XML SQL Utility to insert, update, delete the resulting "canonical" XML for you.

The following built-in XSQL actions make exploiting this capability easy from within your XSQL pages:

- `<xsql:insert-request>`

  Insert the optionally transformed XML document that was posted in the request into a table.Table 9–10 lists the required and optional attributes supported by this action.

- `<xsql:update-request>`

  Update the optionally transformed XML document that was posted in the request into a table or view. Table 9–11 lists the required and optional attributes supported by this action.

- `<xsql:delete-request>`

  Delete the optionally transformed XML document that was posted in the request from a table or view. Table 9–12 lists the required and optional attributes supported by this action.

- `<xsql:insert-param>`

  Insert the optionally transformed XML document that was posted as the value of a request parameter into a table or view. Table 9–13 lists the required and optional attributes supported by this action.

If you target a database view with your insert, then you can create `INSTEAD OF INSERT` triggers on the view to further automate the handling of the posted information. For example, an `INSTEAD OF INSERT` trigger on a view could use PL/SQL to check for the existence of a record and intelligently choose whether to do an `INSERT` or an `UPDATE` depending on the result of this check.

Table 9–10 Attributes for <xsql:insert-request>

Attribute Name	Description
table = "*string*"	Name of the table, view, or synonym to use for inserting the XML information.
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 will be inserted. If supplied, then only these columns will be inserted. If not supplied, all columns will be inserted, with NULL values for columns whose values do not appear in the XML document.
commit-batch-size = "*integer*"	If a positive, nonzero number N is specified, then after each batch of N inserted records, a commit will be issued. Default batch size is zero (0) if not specified, meaning not to commit interim batches.
date-format = "*string*"	Date format mask to use for interpreting date field values in XML being inserted. Valid values are those documented for the `java.text.SimpleDateFormat` class.

Table 9–11 Attributes for <xsql:update-request>

Attribute Name	Description
table = "*string*"	Name of the table, view, or synonym to use for inserting the XML information.

Table 9–11 Attributes for <xsql:update-request>

Attribute Name	Description
key-columns = "*string*"	Space-delimited or comma-delimited list of one or more column names whose values in the posted XML document will be used to identify the existing rows to update.
transform = "*URL*"	Relative or absolute URL of the XSLT transformation to use to transform the document to be inserted into canonical ROWSET/ROW format.
columns = "*string*"	Space-delimited or comma-delimited list of one or more column names whose values will be updated. If supplied, then only these columns will be updated. If not supplied, all columns will be updated, with NULL values for columns whose values do not appear in the XML document.
commit-batch-size = "*integer*"	If a positive, nonzero number N is specified, then after each batch of N inserted records, a commit will be issued. Default batch size is zero (0) if not specified, meaning not to commit interim batches.
date-format = "*string*"	Date format mask to use for interpreting date field values in XML being inserted. Valid values are those documented for the `java.text.SimpleDateFormat` class.

Table 9–12 Attributes for <xsql:delete-request>

Attribute Name	Description
table = "*string*"	Name of the table, view, or synonym to use for inserting the XML information.
key-columns = "*string*"	Space-delimited or comma-delimited list of one or more column names whose values in the posted XML document will be used to identify the existing rows to update.
transform = "*URL*"	Relative or absolute URL of the XSLT transformation to use to transform the document to be inserted into canonical ROWSET/ROW format.
commit-batch-size = "*integer*"	If a positive, nonzero number N is specified, then after each batch of N inserted records, a commit will be issued. Default batch size is zero (0) if not specified, meaning not to commit interim batches.

Table 9–13 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 information.
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 will be inserted. If supplied, then only these columns will be inserted. If not supplied, all columns will be inserted, with NULL values for columns whose values do not appear in the XML document.
commit-batch-size = "*integer*"	If a positive, nonzero number N is specified, then after each batch of N inserted records, a commit will be issued. Default batch size is zero (0) if not specified, meaning not to commit interim batches.
date-format = "*string*"	Date format mask to use for interpreting date field values in XML being inserted. Valid values are those documented for the `java.text.SimpleDateFormat` class.

Understanding Different XML Posting Options

There are three different ways that the XSQL pages framework can handle posted information.

1. A client program can send an HTTP POST message that targets an XSQL page, whose request body contains an XML document and whose HTTP header reports a ContentType of "`text/xml`".

   In this case, you can use the `<xsql:insert-request>`, `<xsql:update-request>`, or the `<xsql:delete-request>` action and the content of the posted XML will be insert, updated, or deleted in the target table as indicated. If you transform the posted XML document using an XSLT transformation, the posted XML document is the source document for this transformation.

2. A client program can send an HTTP GET request for an XSQL page, one of whose parameters contains an XML document.

In this case, you can use the <xsql:insert-param> action and the content of the posted XML parameter value will be inserted in the target table as indicated. If you transform the posted XML document using an XSLT transformation, the XML document in the parameter value is the source document for this transformation.

3. A browser can submit an HTML form with method="POST" whose action targets an XSQL page. In this case, by convention the browser sends an HTTP POST message whose request body contains an encoded version of all of the HTML form's fields and their values with a ContentType of "application/x-www-form-urlencoded"

   In this case, there request does not contain an XML document, but instead an encoded version of the form parameters. However, to make all three of these cases uniform, the XSQL page processor will (on demand) materialize an XML document from the set of form parameters, session variables, and cookies contained in the request. Your XSLT transformation then transforms this dynamically-materialized XML document into canonical form for insert, update, or delete using <xsql:insert>, <xsql:update-request>, or <xsql:delete-request> respectively.

When working with posted HTML forms, the dynamically materialized XML document will have the following form:

```
<request>
  <parameters>
    <firstparamname>firstparamvalue</firstparamname>
      :
    <lastparamname>lastparamvalue</lastparamname>
  </parameters>
  <session>
    <firstparamname>firstsessionparamvalue</firstparamname>
       :
    <lastparamname>lastsessionparamvalue</lastparamname>
  </session>
  <cookies>
    <firstcookie>firstcookievalue</firstcookiename>
        :
    <lastcookie>firstcookievalue</lastcookiename>
  </cookies>
</request>
```

If multiple parameters are posted with the same name, then they will automatically be "row-ified" to make subsequent processing easier. This means, for example, that a request which posts or includes the following parameters:

- `id` = 101
- `name` = Steve
- `id` = 102
- `name` = Sita
- `operation` = update

Will create a "row-ified" set of parameters like:

```
<request>
  <parameters>
    <row>
      <id>101</id>
      <name>Steve</name>
    </row>
    <row>
      <id>102</id>
      <name>Sita</name>
    </row>
    <operation>update</operation>
  </parameters>
       :
</request>
```

Since you will need to provide an XSLT stylesheet that transforms this materialized XML document containing the request parameters into canonical format for your target table, it might be useful to build yourself an XSQL page like this:

```
<!--
 | ShowRequestDocument.xsql
 | Show Materialized XML Document for an HTML Form
 +-->
<xsql:include-request-params xmlns:xsql="urn:oracle-xsql"/>
```

With this page in place, you can temporarily modify your HTML form to post to the `ShowRequestDocument.xsql` page, and in the browser you will see the "raw" XML for the materialized XML request document which you can save out and use to develop the XSLT transformation.

Using Custom XSQL Action Handlers

When you need to perform tasks that are not handled by the built-in action handlers, the XSQL Pages framework allows custom actions to be invoked to do

virtually any kind of job you need done as part of page processing. Custom actions can supply arbitrary XML content to the data page and perform arbitrary processing. See Writing Custom XSQL Action Handlers later in this chapter for more details on writing custom action handlers in Java. Here we explore how to make use of a custom action handler, once it's already created.

To invoke a custom action handler, use the built-in `<xsql:action>` action element. It has a single, required attribute named `handler` whose value is the fully-qualified Java class name of the action you want to invoke. The class must implement the `oracle.xml.xsql.XSQLActionHandler` interface. For example:

```
<xsql:action handler="yourpackage.YourCustomHandler"/>
```

Any number of additional attribute can be supplied to the handler in the normal way. For example, if the `yourpackage.YourCustomHandler` is expecting a attributes named `param1` and `param2`, you use the syntax:

```
<xsql:action handler="yourpackage.YourCustomHandler" param1="xxx" param2="yyy"/>
```

Some action handlers, perhaps in addition to attributes, may expect text content or element content to appear inside the `<xsql:action>` element. If this is the case, simply use the expected syntax like:

```
<xsql:action handler="yourpackage.YourCustomHandler" param1="xxx" param2="yyy">
   Some Text Goes Here
</xsql:action>
```

or this:

```
<xsql:action handler="yourpackage.YourCustomHandler" param1="xxx" param2="yyy">
  <some>
    <other/>
    <elements/>
    <here/>
  </some>
</xsql:action>
```

Description of XSQL Servlet Examples

Figure 9–14 lists the XSQL Servlet example applications supplied with the software in the `./demo` directory.

Table 9–14 XSQL Servlet Examples

Demonstration Name	Description
Hello World `./demo/helloworld`	Simplest possible XSQL page.
Do You XML Site `./demo/doyouxml`	XSQL page shows how a to build a data-driven web site with an XSQL page. Uses SQL, XSQL-substitution variables in queries, and XSLT to format. Uses substitution parameters in SQL statements in `<xsql:query>` tags, and in attributes to `<xsql:query>` tags, to control for example how many records to display, or to skip, for paging through query results.
Employee Page `./demo/emp`	XSQL page displays XML data from EMP table, using XSQL page parameters to control employees and data sorting. Uses an associated XSLT Stylesheet to format results as HTML version of emp.xsql page. This is the form action hence you can fine tune your search criteria.
Insurance Claim Page `./demo/insclaim`	Shows sample queries over a structured, Insurance Claim object view. `insclaim.sql` sets up the INSURANCE_CLAIM_VIEW object view and populates it with sample data.
Invalid Classes Page `./demo/classerr`	XSQL Page uses `invalidclasses.xsl` to format a "live" list of current Java class compilation errors in your schema. The .sql script sets up XSQLJavaClassesView object view for the demo. Master / detail information from object view is formatted into HTML by the `invalidclasses.xsl` stylesheet in the server.
Airport Code Validation `./demo/airport`	XSQL page returns a "datagram" of information about airports based on their three-letter codes. 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. `airport.htm` page demonstrates how to use the XML results of `airport.xsql` page from a web page using JavaScript to exploit built-in XML Document Object Model (DOM) functionality in Internet Explorer 5.0. When you enter the three-letter airport code on the web page, a JavaScript fetches the XML datagram from XSQL Servlet over the web corresponding to the code you entered. If the return indicates no match, the program collects a "picklist" of possible matches based on information returned in the XML "datagram" from XSQL Servlet

Table 9–14 XSQL Servlet Examples (Cont.)

Demonstration Name	Description
Airport Code Display `./demo/airport`	Demonstrates using the same XSQL page as the Airport Code Validation example but supplying an XSLT Stylesheet name in the request. This causes the airport information to be formatted as an HTML form instead of being returned as raw XML.
Emp / Dept Object Demo `./demo/empdept`	How to use an object view to group master / detail information from two existing "flat" tables like EMP and DEPT. `empdeptobjs.sql` script creates the object view and INSTEAD OF INSERT triggers, allowing the use of master / detail view as an insert target of xsql:insert-request. `empdept.xsl` stylesheet illustrates an example of the “simple form” of an XSLT stylesheet that can look just like an HTML page without the extra xsl:stylesheet or xsl:transform at the top. Part of XSLT 1.0 specification called using a Literal Result Element as Stylesheet. Shows how to generate an HTML page that includes the <link rel=”stylesheet”> to allow the generated HTML to fully leverage CSS for centralized HTML style information, found in the `coolcolors.css` file.
Adhoc Query Visualization `./demo/adhocsql`	Shows how to pass an SQL query and XSLT Stylesheet to use as parameters to the server. **NOTE:** *Deploying this demo page to your production environment should be given particular consideration because it allows the results of any SQL query in XML format over the Web that your SCOTT user account has access to.*
XML Document Demo `./demo/document`	How to insert XML documents into relational tables. `docdemo.sql` script creates a user-defined type called `XMLDOCFRAG` containing an attribute of type CLOB. ■ Insert the text of the document in `./xsql/demo/xml99.xml` and provide the name `xml99.xsl` as the stylesheet ■ Insert the text of the document in `./xsql/demo/JDevRelNotes.xml` with the stylesheet `relnotes.xsl`. `docstyle.xsql` page illustrates 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. XML Document demo uses client-side XML features of Internet Explorer 5.0 to check the document for well-formedness before it is posted to the server.

Table 9–14 XSQL Servlet Examples (Cont.)

Demonstration Name	Description
XML Insert Request Demo `./demo/insertxml`	Posts XML from a client to an XSQL Page that inserts the posted XML information into a database table using the `<xsql:insert-request>` action element. The demo accepts XML documents in the moreover.com XML-based news format. The program posting the XML is a client-side web page using Internet Explorer 5.0 and the XMLHttpRequest object from JavaScript. The source for `insertnewsstory.xsql` page, specifies a table name and XSLT Transform name. `moreover-to-newsstory.xsl` stylesheet transforms the incoming XML into canonical format that `OracleXMLSave` utility can insert. Copy and paste the example <article> element several times within the <moreovernews> element to insert several new articles in one shot. `newsstory.sql` 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,....
SVG Demo `./demo/svg`	`deptlist.xsql` page displays a simple list of departments with hyperlinks to `SalChart.xsql` page. `SalChart.xsql` page queries employees for a given department passed in as a parameter and uses the SalChart.xsql *stylesheet* to format the result into a Scalable Vector Graphics drawing, a bar chart comparing salaries of the employees in that department.
PDF Demo `./demo/fop`	`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.

Setting Up the Demo Data

To set up the demo data do the following:

1. Change directory to the ./demo directory on your machine.
2. In this directory, run SQLPLUS. Connect to your database as CTXSYS/CTXSYS — the schema owner for Oracle9*i* Text (Intermedia Text) packages — and issue the command

   ```
   GRANT EXECUTE ON CTX_DDL TO SCOTT;
   ```

3. Connect to your database as SYSTEM/MANAGER and issue the command:

   ```
   GRANT QUERY REWRITE TO SCOTT;
   ```

This allows SCOTT to create a functional index that one of the demos uses to perform case-insensitive queries on descriptions of airports.

4. Connect to your database as SCOTT/TIGER.
5. Run the script `install.sql` in the ./demo directory. This script runs all SQL scripts for all the demos.

```
install.sql
@@insclaim/insclaim.sql
@@document/docdemo.sql
@@classerr/invalidclasses.sql
@@airport/airport.sql
@@insertxml/newsstory.sql
@@empdept/empdeptobjs.sql
```

6. Change directory to ./doyouxml subdirectory, and run the following:

```
imp scott/tiger file=doyouxml.dmp
```

to import sample data for the "Do You XML? Site" demo.

7. To experience the Scalable Vector Graphics (SVG) demonstration, install an SVG plug-in into your browser, such as Adobe SVG Plug-in.

Advanced XSQL Pages Topics

Understanding Client Stylesheet-Override Options

If the current XSQL page being requested allows it, you can supply an XSLT stylesheet URL in the request to override the default stylesheet that would have been used — or to apply a stylesheet where none would have been applied by default. The client-initiated stylesheet URL is provided by supplying the `xml-stylesheet` parameter as part of the request. The valid values for this parameter are:

- Any relative URL, interpreted relative to the XSQL page being processed
- Any absolute URL using the http protocol scheme, provided it references a trusted host (as defined in the `XSQLConfig.xml` file)
- The literal value `none`

This last value, `xml-stylesheet=none`, is particularly useful during development to temporarily "short-circuit" the XSLT stylesheet processing to see

what XML datagram your stylesheet is actually seeing. This can help understand why a stylesheet might not be producing the expected results.

Client-override of stylesheets for an XSQL page can be disallowed either by:

- Setting the `allow-client-style` configuration parameter to `no` in the `XSQLConfig.xml` file, or
- Explicitly including an `allow-client-style="no"` attribute on the document element of any XSQL page

If client-override of stylesheets has been globally disabled by default in the XSQLConfig.xml configuration file, any page can still enable client-override explicitly by including an `allow-client-style="yes"` attribute on the document element of that page.

Controlling How Stylesheets Are Processed

Controlling the Content Type of the Returned Document

Setting the content type of the information you serve is very important. It allows the requesting client to correctly interpret the information that you send back.If your stylesheet uses an `<xsl:output>` element, the XSQL Page Processor infers the media type and encoding of the returned document from the `media-type` and `encoding` attributes of `<xsl:output>`.

For example, the following stylesheet uses the `media-type="application/vnd.ms-excel"` attribute on `<xsl:output>` to transform the results of an XSQL page containing a standard query over the emp table into Microsoft Excel spreadsheet format.

```
<?xml version="1.0"?>
<!-- empToExcel.xsl -->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" media-type="application/vnd.ms-excel"/>
  <xsl:template match="/">
   <html>
     <table>
       <tr><th>EMPNO</th><th>ENAME</th><th>SAL</th></tr>
       <xsl:for-each select="ROWSET/ROW">
         <tr>
           <td><xsl:value-of select="EMPNO"/></td>
           <td><xsl:value-of select="ENAME"/></td>
           <td><xsl:value-of select="SAL"/></td>
         </tr>
```

```
      </xsl:for-each>
    </table>
   </html>
  </xsl:template>
</xsl:stylesheet>
```

An XSQL page that makes use of this stylesheet looks like this:

```
<?xml version="1.0"?>
<?xml-stylesheet href="empToExcel.xsl" type="text/xsl"?>
<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
  select * from emp order by sal desc
</xsql:query>
```

Assigning the Stylesheet Dynamically

As we've seen, if you include an `<?xml-stylesheet?>` processing instruction at the top of your `.xsql` file, it will be considered by the XSQL page processor for use in transforming the resulting XML datagram. For example:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="emp.xsl"?>
<page connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:query>
    SELECT * FROM emp ORDER BY sal DESC
  </xsql:query>
</page>
```

would use the `emp.xsl` stylesheet to transform the results of the EMP query in the server tier, before returning the response to the requestor. The stylesheet is accessed by the relative or absolute URL provided in the `href` pseudo-attribute on the `<?xml-stylesheet?>` processing instruction.

By including one or more parameter references in the value of the `href` pseudo-attribute, you can dynamically determine the name of the stylesheet. For example, this page selects the name of the stylesheet to use from a table by assigning the value of a page-private parameter using a query.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="{@sheet}.xsl"?>
<page connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:set-page-param bind-params="UserCookie" name="sheet">
    SELECT stylesheet_name
      FROM user_prefs
     WHERE username = ?
  </xsql:set-page-param>
```

```
  <xsql:query>
    SELECT * FROM emp ORDER BY sal DESC
  </xsql:query>
</page>
```

Processing Stylesheets in the Client

Some browsers like Microsoft's Internet Explorer 5.0 and higher support processing XSLT stylesheets in the client. These browsers recognize the stylesheet to be processed for an XML document in the same way that a server-side XSQL page does, using an `<?xml-stylesheet?>` processing instruction. This is not a coincidence. The use of `<?xml-stylesheet?>` for this purpose is part of the W3C Recommendation from June 29, 1999 entitled "Associating Stylesheets with XML Documents, Version 1.0"

By default, the XSQL page processor performs XSLT transformations in the server, however by adding on additional pseudo-attribute to your `<?xml-stylesheet?>` processing instruction in your XSQL page — `client="yes"` — the page processor will defer the XSLT processing to the client by serving the XML datagram "raw", with the current `<?xml-stylesheet?>` at the top of the document.

One important point to note is that Internet Explorer 5.0 shipped in late 1998, containing an implementation of the XSL stylesheet language that conformed to a December 1998 Working Draft of the standard. The XSLT 1.0 Recommendation that finally emerged in November of 1999 had significant changes from the earlier working draft version on which IE5 is based. This means that IE5 browsers understand a different "dialect" of XSLT than all other XSLT processors — like the Oracle XSLT processor — which implement the XSLT 1.0 Recommendation syntax.

Toward the end of 2000, Microsoft released version 3.0 of their MSXML components as a Web-downloadable release. This latest version *does* implement the XSLT 1.0 standard, however in order for it to be used as the XSLT processor inside the IE5 browser, the user must go through additional installation steps. Unfortunately there is no way for a server to detect that the IE5 browser has installed the latest XSLT components, so until the Internet Explorer 6.0 release emerges — which will contain the latest components by default and which will send a detectably different User-Agent string containing the 6.0 version number — stylesheets delivered for client processing to IE5 browsers should use the earlier IE5-"flavor" of XSL.

What we need is a way to request that an XSQL page use different stylesheets depending on the User-Agent making the request. Luckily, the XSQL Pages framework makes this easy and we learn how in the next section.

Providing Multiple, UserAgent-Specific Stylesheets

You can include multiple `<?xml-stylesheet?>` processing instructions at the top of an XSQL page and any of them can contain an optional `media` pseudo-attribute. If specified, the `media` pseudo-attribute's value is compared case-insensitively with the value of the HTTP header's User-Agent string. If the value of the `media` pseudo-attribute matches a part of the User-Agent string, then the processor selects the current `<?xml-stylesheet?>` processing instruction for use, otherwise it ignores it and continues looking. The first matching processing instruction in document order will be used. A processing instruction *without* a `media` pseudo-attribute matches all user agents so it can be used as the fallback / default.

For example, the following processing instructions at the top of an .xsql file...

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" media="lynx" href="doyouxml-lynx.xsl" ?>
<?xml-stylesheet type="text/xsl" media="msie 5" href="doyouxml-ie.xsl" ?>
<?xml-stylesheet type="text/xsl" href="doyouxml.xsl" ?>
<page xmlns:xsql="urn:oracle-xsql" connection="demo">
  :
```

will use `doyouxml-lynx.xsl` for Lynx browsers, `doyouxml-ie.xsl` for Internet Explorer 5.0 or 5.5 browsers, and `doyouxml.xsl` for all others.

Table 9–15 summarizes all of the supported pseudo-attributes allowed on the `<?xml-stylesheet?>` processing instruction.

Table 9–15 Pseudo-Attributes for <?xml-stylesheet?>

Attribute Name	Description
type = "*string*"	Indicates the 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 should 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 `XSQLConfig.xml` file.
media = "*string*"	This attribute is optional. If provided, its value is used to perform a case-*insensitive* match on the `User-Agent` string from the HTTP header sent by the requesting device. The current `<?xml-stylesheet?>` processing instruction will only be used if the `User-Agent` string contains the value of the `media` attribute, otherwise it is ignored.

Table 9–15 Pseudo-Attributes for <?xml-stylesheet?>

Attribute Name	Description
client = "*boolean*"	If set to `yes`, caused the XSQL page processor to defer the processing of the associated XSLT stylesheet to the client. The "raw" XML datagram will be sent to the client with the current `<?xml-stylesheet?>` processing instruction at the top of the document. The default if not specified is to perform the transform in the server.
serializer = "*string*"	By default, the XSQL page processor uses the: ■ XML DOM serializer if no XSLT stylesheet is used ■ XSLT processor's serializer, if XSLT stylesheet is used Specifying this pseudo-attribute indicates that a custom serializer implementation should be used instead. Valid values are *either* the name of a custom serializer defined in the `<serializerdefs>` section of the `XSQLConfig.xml` file, or the string `java:`*`fully.qualified.Classname`*. If both an XSLT stylesheet and the serializer attribute are present, then the XSLT transform is performed first, then the custom serializer is invoked to render the final result to the OutputStream or PrintWriter.

Using XSQLConfig.xml to Tune Your Environment

Use the `XSQLConfig.xml` File to tune your XSQL pages environment. Table 9–16 defines all of the parameters that can be set.

Table 9–16 XSQLConfig.xml Configuation Settings

Configuration Setting Name
`XSQLConfig/servlet/output-buffer-size`
Sets the size (in bytes) of the buffered output stream. If your servlet engine already buffers I/O to the Servlet Output Stream, then you can set to `0` to avoid additional buffering. Default value is `0`. Valid value is any non-negative integer.

Table 9–16 XSQLConfig.xml Configuation Settings

Configuration Setting Name
XSQLConfig/servlet/suppress-mime-charset/media-type The XSQL Servlet sets the HTTP ContentType header to indicate the MIME type of the resource being returned to the request. By default, the XSQL Servlet includes the optional character set information in the MIME type. For a particular MIME type, you can suppress the inclusion of the character set information by including a <media-type> element, with the desired MIME type as its contents. You may list any number of <media-type> elements. Valid value is any string.
XSQLConfig/processor/character-set-conversion/default-charset By default, the XSQL page processor does charater set conversion on the value of HTTP parameters to compensate for the default character set used by most servlet engines. The default base character set used for conversion is the Java character set 8859_1 corresponding to IANA's ISO-8859-1 character set. If your servlet engine uses a different character set as its base character set you can now specify that value here. To suppress character set conversion, specify the empty element <none/> as the content of the <default-charset> element, instead of a character set name. This is useful if you are working with parameter values that are correctly representable using your servlet's default character set, and eliminates a small amount of overhead associated with performing the character set conversion. Valid values are any Java character set name, or the element <none/>.
XSQLConfig/processor/reload-connections-on-error Connection definitions are cached when the XSQL Page Processor is initialized. Set this setting to yes to cause the processor to reread the XSQLConfig.xml file to reload connection definitions if an attempt is made to request a connection name that's not in the cached connection list. The yes setting is useful during development when you might be adding new <connection> definitions to the file while the servlet is running. Set to no to avoid reloading the connection definition file when a connection name is not found in the in-memory cache. Default is yes. Valid values are yes and no.
XSQLConfig/processor/default-fetch-size Sets the default value of the row fetch size for retrieving information from SQL queries from the database. Only takes effect if you are using the Oracle JDBC Driver, otherwise the setting is ignored. Useful for reducing network round-trips to the database from the servlet engine running in a different tier. Default is 50. Valid value is any nonzero positive integer.

Table 9–16 XSQLConfig.xml Configuation Settings

Configuration Setting Name
`XSQLConfig/processor/page-cache-size` Sets the size of the XSQL cache for XSQL page templates. This determines the maximum number of XSQL pages that will be cached. Least recently used pages get "bumped" out of the cache if you go beyond this number. Default is `25`. Valid value is any nonzero positive integer.
`XSQLConfig/processor/stylesheet-cache-size` Sets the size of the XSQL cache for XSLT stylesheets. This determines the maximum number of stylesheets that will be cached. Least recently used stylesheets get "bumped" out of the cache if you go beyond this number. Default is `25`. Valid value is any nonzero positive integer.
`XSQLConfig/processor/stylesheet-pool/initial` Each cached stylesheet is actually a pool of cached stylesheet instances to improve throughput. Sets the initial number of stylesheets to be allocated in each stylesheet pool. Default is `1`. Valid value is any nonzero positive integer.
`XSQLConfig/processor/stylesheet-pool/increment` Sets the number of stylesheets to be allocated when the stylesheet pool must grow due to increased load on the server. Default is `1`. Valid value is any nonzero positive integer.
`XSQLConfig/processor/stylesheet-pool/timeout-seconds` Sets the number of seconds of inactivity that must transpire before a stylesheet instance in the pool will be removed to free resources as the pool tries to "shrink" back to its initial size. Default is `60`. Valid value is any nonzero positive integer.
`XSQLConfig/processor/connection-pool/initial` The XSQL page processor's default connection manager implements connection pooling to improve throughput. This setting controls the initial number of JDBC connections to be allocated in each connection pool. Default is `2`. Valid value is any nonzero positive integer.
`XSQLConfig/processor/connection-pool/increment` Sets the number of connections to be allocated when the connection pool must grow due to increased load on the server. Default is `1`. Valid value is any nonzero positive integer.

Table 9–16 XSQLConfig.xml Configuation Settings

Configuration Setting Name
XSQLConfig/processor/connection-pool/timeout-seconds Sets the number of seconds of inactivity that must transpire before a JDBC connection in the pool will be removed to free resources as the pool tries to "shrink" back to its initial size. Default is 60. Valid value is any 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, this setting defaults to oracle.xml.xsql.XSQLConnectionManagerFactoryImpl. Default is oracle.xml.xsql.XSQLConnectionManagerFactoryImpl. Valid value is any class name that implements the oracle.xml.xsql.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 or TABLE, and the default if not specified is CLOB. If set to CLOB, the processor uses temporary CLOB to retrieve the OWA page buffer. If set to TABLE the processor uses a more efficient approach that requires the existence of the Oracle user-defined type named XSQL_OWA_ARRAY which must be created by hand using the DDL statement: `CREATE TYPE xsql_owa_array AS TABLE OF VARCHAR2(32767)`
XSQLConfig/processor/timing/page Determines whether a the XSQL page processor adds an xsql-timing attribute to the document element of the page whose value reports the elapsed number of milliseconds required to process the page. Default is no. Valid value is yes or no.
XSQLConfig/processor/timing/action Determines whether a the XSQL page processor adds comment to the page just before the action element whose contents reports the elapsed number of milliseconds required to process the action. Default is no. Valid value is yes or no.

Table 9–16 XSQLConfig.xml Configuation Settings

Configuration Setting Name
XSQLConfig/processor/security/stylesheet/defaults/allow-client-style
While developing an application, it is frequently useful to take advantage of the XSQL page processor's per-request stylesheet override capability by providing a value for the special `xml-stylesheet` parameter in the request. One of the most common uses is to provide the `xml-stylesheet=none` combination to temporarily disable the application of the stylesheet to "peek" underneath at the raw XSQL data page for debugging purposes. When development is completed, you could explicitly add the `allow-client-style="no"` attribute to the document element of each XSQL page to prohibit client overriding of the stylesheet in the production application. However, using this configuration setting, you can globally change the default behavior for `allow-client-style` in a single place. Note that this only provides the *default* setting for this behavior. If the `allow-client-style="yes|no"` attribute is explicitly specified on the document element for a given XSQL page, its value takes precedence over this global default. Valid values are `yes` and `no`.
XSQLConfig/processor/security/stylesheet/trusted-hosts/host
XSLT stylesheets can invoke extension functions. In particular, the Oracle XSLT processor — which the XSQL page processor uses to process all XSLT stylesheets — supports *Java* extension functions. Typically your XSQL pages will refer to XSLT stylesheets using relative URL's The XSQL page processor enforces that any absolute URL to an XSLT stylesheet that is processed must be from a trusted host whose name is listed here in the configuration file. You may list any number of `<host>` elements inside the `<trusted-hosts>` element. The name of the local machine, `localhost`, and `127.0.0.1` are considered trusted hosts by default. Valid values are any hostname or IP address.
XSQLConfig/http/proxyhost
Sets the name of the HTTP proxy server to use when processing URL's with the http protcol scheme. Valid value is any hostname or IP address.
XSQLConfig/http/proxyport
Sets the port number of the HTTP proxy server to use when processing URL's with the http protcol scheme. Valid value is any nonzero integer.

Table 9–16 XSQLConfig.xml Configuation Settings

Configuration Setting Name
XSQLConfig/connectiondefs/connection Defines a "nickname" and the JDBC connection details for a named connection for use by the XSQL page processor. You may supply any number of <connection> element children of <connectiondefs>. Each connection definition must supply a name attribute, and may supply appropriate children elements <username>, <password>, <driver>, <dburl>, and <autocommit>.
XSQLConfig/connectiondefs/connection/username Defines the username for the current connection.
XSQLConfig/connectiondefs/connection/password Defines the password for the current connection.
XSQLConfig/connectiondefs/connection/dburl Defines the JDBC connection URL for the current connection.
XSQLConfig/connectiondefs/connection/driver Specifies the fully-qualified Java class name of the JDBC driver to be used for the current connection. If not specified, defaults to oracle.jdbc.driver.OracleDriver.
XSQLConfig/connectiondefs/connection/autocommit Explicity sets the Auto Commit flag for the current connection. If not specified, connection uses JDBC driver's default setting for Auto Commit.
XSQLConfig/serializerdefs/serializer Defines a named custom serializer implementation. You may supply any number of <serializer> element children of <serializerdefs>. Each must specify both a <name> and a <class> child element.
XSQLConfig/serializerdefs/serializer/name Defines the name of the current custom serializer definition.
XSQLConfig/connectiondefs/connection/class Specifies the fully-qualified Java class name of the current custom serializer. The class must implement the oracle.xml.xsql.XSQLDocumentSerializer interface.

Using the FOP Serializer to Produce PDF Output

Using the XSQL Pages framework's support for custom serializers, the oracle.xml.xsql.serializers.XSQLFOPSerializer is provided for integrating with the Apache FOP processor (http://xml.apache.org/fop). The FOP

processor renders a PDF document from an XML document containing XSL Formatting Objects (`http://www.w3.org/TR/xsl`).

For example, given the following XSLT stylesheet, `EmpTableFO.xsl`:

```
<?xml version="1.0"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format" xsl:version="1.0"
         xmlns:xsl="http://www.w3.org/1999/XSL/Transform">


  <!-- defines the layout master -->
  <fo:layout-master-set>
    <fo:simple-page-master master-name="first"
                           page-height="29.7cm"
                           page-width="21cm"
                           margin-top="1cm"
                           margin-bottom="2cm"
                           margin-left="2.5cm"
                           margin-right="2.5cm">
      <fo:region-body margin-top="3cm"/>
    </fo:simple-page-master>
  </fo:layout-master-set>

  <!-- starts actual layout -->
  <fo:page-sequence master-reference="first">

  <fo:flow flow-name="xsl-region-body">

      <fo:block font-size="24pt" line-height="24pt" font-weight="bold"
start-indent="15pt">
        Total of All Salaries is $<xsl:value-of select="sum(/ROWSET/ROW/SAL)"/>
      </fo:block>

      <!-- Here starts the table -->
      <fo:block border-width="2pt">
        <fo:table>
          <fo:table-column column-width="4cm"/>
          <fo:table-column column-width="4cm"/>
          <fo:table-body font-size="10pt" font-family="sans-serif">
            <xsl:for-each select="ROWSET/ROW">
              <fo:table-row line-height="12pt">
                <fo:table-cell>
                  <fo:block><xsl:value-of select="ENAME"/></fo:block>
                </fo:table-cell>
                <fo:table-cell>
```

```
                <fo:block><xsl:value-of select="SAL"/></fo:block>
              </fo:table-cell>
            </fo:table-row>
          </xsl:for-each>
        </fo:table-body>
      </fo:table>
    </fo:block>
  </fo:flow>
 </fo:page-sequence>
</fo:root>
```

Note: To use the XSQL FOP Serializer, you need to add these additional Java archives to your server's CLASSPATH:

- `xsqlserializers.jar` **— supplied with Oracle XSQL**
- `fop.jar` **— From Apache, version 0.16 or higher**
- `w3c.jar` **— from the FOP distribution's** `./lib` **directory**

Using XSQL Page Processor Programmatically

The `XSQLRequest` class, allows you to utilize the XSQL page processor "engine" from within your own custom Java programs. Using the API is simple. You construct an instance of `XSQLRequest`, passing the XSQL page to be processed into the constructor as one of the following:

- `String` containing a URL to the page
- `URL` object for the page
- In-memory `XMLDocument`

Then you invoke one of the following methods to process the page:

- `process()`— to write the result to a PrintWriter or OutputStream, or
- `processToXML()` — to return the result as an XML Document

If you want to use the built-in XSQL Connection Manager — which implements JDBC connection pooling based on `XSQLConfig.xml`-based connection definitions — then the XSQL page is all you need to pass to the constructor. Optionally, you can pass in a custom implementation for the `XSQLConnectionManagerFactory` interface as well, if you want to use your own connection manager implementation.

Note that the ability to pass the XSQL page to be processed as an in-memory XML Document object means that you can dynamically generate any valid XSQL page for processing using any means necessary, then pass the page to the XSQL engine for evaluation.

When processing a page, there are two additional things you may want to do as part of the request:

- Pass a set of parameters to the request

  You accomplish this by passing any object that implements the `Dictionary` interface, to the `process()` or `processToXML()` methods. Passing a `HashTable` containing the parameters is one popular approach.

- Set an XML document to be processed by the page as if it were the "posted XML" message body

  You can do this using the `setPostedDocument()` method on the XSQLRequest object.

Here is a simple example of processing a page using `XSQLRequest`:

```
import oracle.xml.xsql.XSQLRequest;
import java.util.Hashtable;
import java.io.PrintWriter;
import java.net.URL;
public class XSQLRequestSample {
  public static void main( String[] args) throws Exception {
     // Construct the URL of the XSQL Page
   URL pageUrl = new URL("file:///C:/foo/bar.xsql");
   // Construct a new XSQL Page request
   XSQLRequest req = new XSQLRequest(pageUrl);
   // Setup a Hashtable of named parameters to pass to the request
   Hashtable params = new Hashtable(3);
   params.put("param1","value1");
   params.put("param2","value2");
   /* If needed, treat an existing, in-memory XMLDocument as if
   ** it were posted to the XSQL Page as part of the request
   req.setPostedDocument(myXMLDocument);
   **
   */
   // Process the page, passing the parameters and writing the output
   // to standard out.
   req.process(params,new PrintWriter(System.out)
                      ,new PrintWriter(System.err));
  }
}
```

Writing Custom XSQL Action Handlers

When the task at hand requires custom processing, and none of the built-in actions does exactly what you need, you can augment your repertoire by writing your own actions that any of your XSQL pages can use.

The XSQL page processor at its very core is an engine that processes XML documents containing "action elements". The page processor engine is written to support any action that implements the `XSQLActionHandler` interface. All of the built-in actions implement this interface.

The XSQL Page Processor processes the actions in a page in the following way. For each action in the page, the engine:

1. Constructs an instance of the action handler class using the default constructor
2. Initializes the handler instance with the action element object and the page processor context by invoking the method:

   ```
   init(Element actionElt,XSQLPageRequest context)
   ```

3. Invokes the method that allows the handler to handle the action:

   ```
   handleAction (Node result)
   ```

For built-in actions, the engine knows the mapping of XSQL action element name to the Java class that implements the action's handler. Table 9–17 lists that mapping explicitly for your reference. For user-defined actions, you use the built-in:

```
<xsql:action handler="fully.qualified.Classname" ... />
```

action whose `handler` attribute provides the fully-qualified name of the Java class that implements the custom action handler.

Table 9–17 Built-In XSQL Elements and Action Handler Classes

XSQL Action Element	Handler Class in oracle.xml.xsql.actions
`<xsql:query>`	`XSQLQueryHandler`
`<xsql:dml>`	`XSQLDMLHandler`
`<xsql:set-stylesheet-param>`	`XSQLStylesheetParameterHandler`
`<xsql:insert-request>`	`XSQLInsertRequestHandler`
`<xsql:include-xml>`	`XSQLIncludeXMLHandler`
`<xsql:include-request-params>`	`XSQLIncludeRequestHandler`
`<xsql:include-posted-xml>`	`XSQLIncludePostedXMLHandler`

Table 9–17 Built-In XSQL Elements and Action Handler Classes

XSQL Action Element	Handler Class in oracle.xml.xsql.actions
<xsql:include-xsql>	XSQLIncludeXSQLHandler
<xsql:include-owa>	XSQLIncludeOWAHandler
<xsql:action>	XSQLExtensionActionHandler
<xsql:ref-cursor-function>	XSQLRefCursorFunctionHandler
<xsql:include-param>	XSQLGetParameterHandler
<xsql:set-session-param>	XSQLSetSessionParamHandler
<xsql:set-page-param>	XSQLSetPageParamHandler
<xsql:set-cookie>	XSQLSetCookieHandler
<xsql:insert-param>	XSQLInsertParameterHandler
<xsql:update-request>	XSQLUpdateRequestHandler
<xsql:delete-request>	XSQLDeleteRequestHandler

Writing your Own Action Handler

To create a custom Action Handler, you need to provide a class that implements the `oracle.xml.xsql.XSQLActionHandler` interface. Most custom action handlers should extend `oracle.xml.xsql.XSQLActionHandlerImpl` that provides a default implementation of the `init()` method and offers a set of useful helper methods that will prove very useful.

When an action handler's handleAction method is invoked by the XSQL page processor, the action implementation gets passed the root node of a DOM Document Fragment to which the action handler should append any dynamically created XML content that should be returned to the page.

The XSQL Page Processor conceptually replaces the action element in the XSQL page *template* with the content of this Document Fragment. It is completely legal for an Action Handler to append *nothing* to this document fragment, if it has no XML content to add to the page.

While writing you custom action handlers, several methods on the XSQLActionHandlerImpl class are worth noting because they make your life a lot easier. Table 9–18 lists the methods that will likely come in handy for you.

Table 9–18 Helpful Methods on oracle.xml.xsql.SQLActionHandlerImpl

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.
getPageRequest	Returns the current XSQL page processor context. Using this object you can then do things like: ■ `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 addition Servlet-specific methods like `getHttpServletRequest`, `getHttpServletResponse`, and `getServletContext`
getAttributeAllowingParam	Retrieve the attribute value from an element, resolving any XSQL lexical parameter references that might appear in the attribute's value. Typically this method is applied to the action element itself, but it is also useful for accessing attributes of any of its sub-elements. To access an attribute value without allowing lexical parameters, use the standard `getAttribute()` method on the DOM Element interface.

Table 9–18 Helpful Methods on oracle.xml.xsql.SQLActionHandlerImpl

Method Name	Description
appendSecondaryDocument	Append the entire contents of an external XML document to the root of the action handler result content.
addResultElement	Simplify appending a single element with text content to the root of the action handler result content.
firstColumnOfFirstRow	Return the first column value of the first row of a SQL statement passed in. Requires the current page to have a connection attribute on its document element, or an error is returned.
bindVariableCount	Returns the number of tokens in the space-delimited list of `bind-params`, indicating how many bind variables are expected to be bound to parameters.
handleBindVariables	Manage 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 already is using a number of bind variables prior to call this method, you can pass the number of existing bind variable "slots" in use as well.
reportErrorIncludingStatement	Report an error, including the offending (SQL) statement that caused the problem, optionally including a numeric error code.
reportFatalError	Report a fatal error.
reportMissingAttribute	Report an error that a required action handler attribute is missing using the standard `<xsql-error>` element.
reportStatus	Report action handler status using the standard `<xsql-status>` element.
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 which might determine its default value.

The following example shows a custom action handler `MyIncludeXSQLHandler` that leverages one of the built-in action handlers and then uses arbitrary Java code to modify the resulting XML fragment returned by that handler before appending its result to the XSQL page:

```
import oracle.xml.xsql.*;
import oracle.xml.xsql.actions.XSQLIncludeXSQLHandler;
import org.w3c.dom.*;
import java.sql.SQLException;
public class MyIncludeXSQLHandler extends XSQLActionHandlerImpl {
  XSQLActionHandler nestedHandler = null;
  public void init(XSQLPageRequest req, Element action) {
    super.init(req, action);
    // Create an instance of an XSQLIncludeXSQLHandler
    // and init() the handler by passing the current request/action
    // This assumes the XSQLIncludeXSQLHandler will pick up its
    // href="xxx.xsql" attribute from the current action element.
    nestedHandler = new XSQLIncludeXSQLHandler();
    nestedHandler.init(req,action);
  }
  public void handleAction(Node result) throws SQLException {
    DocumentFragment df=result.getOwnerDocument().createDocumentFragment();
    nestedHandler.handleAction(df);
    // Custom Java code here can work on the returned document fragment
    // before appending the final, modified document to the result node.
    // For example, add an attribute to the first child
    Element e = (Element)df.getFirstChild();
    if (e != null) {
      e.setAttribute("ExtraAttribute","SomeValue");
    }
    result.appendChild(df);
  }
}
```

If you create custom action handlers that need to work differently based on whether the page is being requested through the XSQL Servlet, the XSQL Command-line Utility, or programmatically through the XSQLRequest class, then in your Action Handler implementation you can call `getPageRequest()` to get a reference to the XSQLPageRequest interface for the current page request. By calling `getRequestType()` on the XSQLPageRequest object, you can see if the request is coming from the "Servlet", "Command Line", or "Programmatic" routes respectively. If the return value is "Servlet", then you can get access to the HTTP Servlet's request, response, and servlet context objects by doing:

```
XSQLServletPageRequest xspr = (XSQLServletPageRequest)getPageRequest();
if (xspr.getRequestType().equals("Servlet")) {
  HttpServletRequest     req  = xspr.getHttpServletRequest();
  HttpServletResponse   resp  = xspr.getHttpServletResponse();
  ServletContext        cont  = xspr.getServletContext();
  // do something fun here with req, resp, or cont however
```

```
  // writing to the response directly from a handler will
  // produce unexpected results. Allow the XSQL Servlet
  // or your custom Serializer to write to the servlet's
  // response output stream at the write moment later when all
  // action elements have been processed.
}
```

Writing Custom XSQL Serializers

You can provide a user-defined serializer class to programmatically control how the final XSQL datapage's XML document should be serialized to a text or binary stream. A user-defined serializer must implement the `oracle.xml.xsql.XSQLDocumentSerializer` interface which comprises the single method:

```
void serialize(org.w3c.dom.Document doc, XSQLPageRequest env) throws Throwable;
```

In this release, DOM-based serializers are supported. A future release may support SAX2-based serializers as well. A custom serializer class is expected to perform the following tasks in the correct order:

1. Set the content type of the serialized stream before writing any content to the output `PrintWriter` (or `OutputStream`).

   You set the type by calling `setContentType()` on the `XSQLPageRequest` that is passed to your serializer. When setting the content type, you can either set just a MIME type like this:

   ```
   env.setContentType("text/html");
   ```

   or a MIME type with an explicit output encoding character set like this:

   ```
   env.setContentType("text/html;charset=Shift_JIS");
   ```

2. Call `getWriter()` or `getOutputStream()` — but not both! — on the `XSQLPageRequest` to get the appropriate `PrintWriter` or `OutputStream` respectively to use for serializing the content.

For example, the following custom serializer illustrates a simple implementation which simply serializes an HTML document containing the name of the document element of the current XSQL data page:

```
package oracle.xml.xsql.serializers;
import org.w3c.dom.Document;
import java.io.PrintWriter;
import oracle.xml.xsql.*;
```

```
public class XSQLSampleSerializer implements XSQLDocumentSerializer {
  public void serialize(Document doc, XSQLPageRequest env) throws Throwable {
    String encoding = env.getPageEncoding();  // Use same encoding as XSQL page
                                              // template. Set to specific
                                              // encoding if necessary
    String mimeType = "text/html"; // Set this to the appropriate content type
    // (1) Set content type using the setContentType on the XSQLPageRequest
    if (encoding != null && !encoding.equals("")) {
      env.setContentType(mimeType+";charset="+encoding);
    }
    else {
      env.setContentType(mimeType);
    }
    // (2) Get the output writer from the XSQLPageRequest
    PrintWriter e = env.getWriter();
    // (3) Serialize the document to the writer
    e.println("<html>Document element is <b>"+
              doc.getDocumentElement().getNodeName()+
              "</b></html>");
  }
}
```

There are two ways to use a custom serializer, depending on whether you need to first perform an XSLT transformation before serializing or not. To perform an XSLT transformation before using a custom serializer, simply add the `serializer="java:`*`fully.qualified.ClassName`*`"` in the `<?xml-stylesheet?>` processing instruction at the top of your page like this:

```
<?xml version="1.0?>
<?xml-stylesheet type="text/xsl" href="mystyle.xsl"
                 serializer="java:my.pkg.MySerializer"?>
```

If you only need the custom serializer, simply leave out the `type` and `href` attributes like this:

```
<?xml version="1.0?>
<?xml-stylesheet serializer="java:my.pkg.MySerializer"?>
```

You can also assign a short nickname to your custom serializers in the `<serializerdefs>` section of the `XSQLConfig.xml` file and then use the nickname (case-sensitive) in the serializer attribute instead to save typing. For example, if you have the following in `XSQLConfig.xml`:

```
<XSQLConfig>
  <!-- etc. -->
  <serializerdefs>
    <serializer>
      <name>Sample</name>
      <class>oracle.xml.xsql.serializers.XSQLSampleSerializer</class>
    </serializer>
    <serializer>
      <name>FOP</name>
      <class>oracle.xml.xsql.serializers.XSQLFOPSerializer</class>
    </serializer>
  </serializerdefs>
</XSQLConfig>
```

then you can use the nicknames "Sample" and / or "FOP" as shown in the following examples:

```
<?xml-stylesheet type="text/xsl" href="emp-to-xslfo.xsl" serializer="FOP"?>
```

or

```
<?xml-stylesheet serializer="Sample"?>
```

The `XSQLPageRequest` interface supports both a `getWriter()` and a `getOutputStream()` method. Custom serializers can call `getOutputStream()` to return an `OutputStream` instance into which binary data (like a dynamically produced GIF image, for example) can be serialized. Using the XSQL Servlet, writing to this output stream results in writing the binary information to the servlet's output stream.

For example, the following serializer illustrates an example of writing out a dynamic GIF image. In this example the GIF image is a static little "ok" icon, but it shows the basic technique that a more sophisticated image serializer would need to use:

```
package oracle.xml.xsql.serializers;
import org.w3c.dom.Document;
import java.io.*;
import oracle.xml.xsql.*;

public class XSQLSampleImageSerializer implements XSQLDocumentSerializer {
   // Byte array representing a small "ok" GIF image
   private static byte[] okGif =
     {(byte)0x47,(byte)0x49,(byte)0x46,(byte)0x38,
      (byte)0x39,(byte)0x61,(byte)0xB,(byte)0x0,
```

```
      (byte)0x9,(byte)0x0,(byte)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();
  }
}
```

Using the XSQL Command-line utility, the binary information is written to the target output file. Using the XSQLRequest programmatic API, two constructors exist that allow the caller to supply the target OutputStream to use for the results of page processing.

Note that your serializer must either call `getWriter()` (for textual output) or `getOutputStream()` (for binary output) but not both. Calling both in the same request will raise an error.

Writing Custom XSQL Connection Managers

You can provide a custom connection manager to replace the built-in connection management mechanism. To provide a custom connection manager implementation, you must provide:

1. A connection manager *factory* object that implements the `oracle.xml.xsql.XSQLConnectionManagerFactory` interface.
2. A connection manager object that implements the `oracle.xml.xsql.XSQLConnectionManager` interface.

Your custom connection manager factory can be set to be used as the default connection manager factory by providing the classname in the `XSQLConfig.xml` file in the section:

```
<!--
 | Set the name of the XSQL Connection Manager Factory
 | implementation. The class must implement the
 | oracle.xml.xsql.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, a custom connection factory can be associated with any individual `XSQLRequest` object using API's provided.

The responsibility of the `XSQLConnectionManagerFactory` is to return an instance of an `XSQLConnectionManager` for use by the current request. In a multithreaded environment like a servlet engine, it is the responsibility of the `XSQLConnectionManager` object to insure that a single `XSQLConnection` instance is not used by two different threads. This can be assured by marking the connection as "in use" for the span of time between the invocation of the `getConnection()` method and the `releaseConnection()` method. The default XSQL connection manager implementation automatically pools named connections, and adheres to this thread-safe policy.

If your custom implementation of `XSQLConnectionManager` implements the optional `oracle.xml.xsql.XSQLConnectionManagerCleanup` interface as well, then your connection manager will be given a chance to cleanup 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, this will give the connection manager a chance to clean up resources as part of the servlet destruction process.

Formatting XSQL Action Handler Errors

Errors raised by the processing of any XSQL Action Elements are reported as XML elements in a uniform way so that XSL Stylesheets can detect their presence and optionally format them for presentation.

The action element in error will be replaced in the page by:

```
<xsql-error action="xxx">
```

Depending on the error the <xsql-error> element contains:

- A nested <message> element
- A <statement> element with the offending SQL statement

Displaying Error Information on Screen

Here is an example of an XSLT stylesheet that uses this information to display error information on the screen:

```
<xsl:if test="//xsql-error">
     <table style="background:yellow">
        <xsl:for-each select="//xsql-error">
           <tr>
            <td><b>Action</b></td>
            <td><xsl:value-of select="@action"/></td>
            </tr>
            <tr valign="top">
            <td><b>Message</b></td>
            <td><xsl:value-of select="message"/></td>
           </tr>
         </xsl:for-each>
     </table>
</xsl:if>
```

XSQL Servlet Limitations

XSQL Servlet has the following limitations:

HTTP Parameters with Multibyte Names

HTTP parameters with multibyte names, for example, a parameter whose name is in Kanji, are properly handled when they are inserted into your XSQL page using <xsql:include-request-params>. An attempt to refer to a parameter with a multibyte name inside the query statement of an <xsql:query> tag will return an empty string for the parameter's value.

Workaround

As a workaround use a non-multibyte parameter name. The parameter can still have a multibyte value which can be handled correctly.

CURSOR() Function in SQL Statements

If you use the CURSOR() function in SQL statements you may get an "Exhausted ResultSet" error if the CURSOR() statements are nested and if the first row of the query returns an empty result set for its CURSOR() function.

Frequently Asked Questions About the XSQL Servlet

This section lists XSQL Servlet questions and answers.

Can I Specify a DTD While Transforming XSQL Output to a WML Document?

I am trying to write my own stylesheet for transforming XSQL output to WML and VML format. These programs, which are mobile phone simulators need a WML document with a specific DTD assigned.

Is there any way to specify a particular DTD while transforming XSQL's output to a WML document?

Answer: Sure. The way you do it is using a built-in facility of the XSLT stylesheet called `<xsl:output>`. Here is an example:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output type="xml" doctype-system="your.dtd"/>
  <xsl:template match="/">
  </xsl:template>
     :
     :
</xsl:stylesheet>
```

This will produce an XML result with the following code in the result:

```
<!DOCTYPE xxxx SYSTEM "your.dtd">
```

where `"your.dtd"` can be any valid absolute or relative URL.

Can I Write XSQL Servlet Conditional Statements?

Is it possible to write conditional statements in an XSQL file? If yes, then what is the syntax to do that?

For example:

```
<xsql:choose>
    <xsql:when test="@security='admin'">
      <xsql:query>
```

```
            SELECT ....
        </xsql:query>
    </xsq:when>
    <xsql:when test="@security='user'">
        <xsql:query>
            SELECT ....
        </xsql:query>
    </xsql:when>
</xsql:if>
```

Answer: Use `<xsql:ref-cursor-function>` to call a PL/SQL procedure that would conditionally return a `REF CURSOR` to the appropriate query.

Can I Use a Value Retrieved in One Query in Another Query's Where Clause?

I have two queries in an XSQL file.

```
<xsql:query>
   select col1,col2
   from table1
</xsql:query>
<xsql:query>
   select col3,col4 from table2
   where col3 = {@col1}    => the value of col1 in the previous query
</xsql:query>
```

How can I use, in the second query, the value of a select list item of the first query?

Answer: You do this with page parameters. Refer to the following example:

```
<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>
```

Can I Use the XSQL Servlet with Non-Oracle Databases?

Can the XSQL Servlet connect to any database that has JDBC support?

Answer: Yes. Just indicate the appropriate JDBC driver class and connection URL in the `XSQLConfig.xml` file's connection definition. Of course, object/relational functionality only works when using Oracle with the Oracle JDBC driver.

How Do I Use the XSQL Servlet to Access the JServ Process?

I am running the demo `helloworld.xsql`. Initially I was getting the following error:

```
XSQL-007 cannot aquire a database connection to process page
```

Now my request times out and I see the following message in the `jserv/log/jserv.log` file:

```
Connections from Localhost/127.0.0.1 are not allowed
```

Is this a security issue? Do we have to give explicit permission to process an XSQL page? If so, how do we do that? I am using Apache Web server and Apache jserver, with Oracle9*i* as the database. I have Oracle client installed and the `Tnsnames.ora` file configured to get database connection. My `XSQconnections.xml` file is configured correctly.

Answer: This looks like a generic JServ problem. You have to make sure that your `security.allowedAddresses=property` in `jserv.properties` allows your current host access to the JServ process where Java runs. It may be helpful to test whether you can successfully run *any* JServ servlet.

How Do I Run XSQL on Oracle8*i* Lite?

I am trying to use XSQL with Oracle8*i* Lite on Windows 98, and the Apache JServ Web server. I am getting the error message `no oljdbc40 in java.library.path`, even though I have set the `olite40.jar` in my `classpath` (which contains the `POLJDBC` driver). Is there anything extra I need to do to run XSQL for Oracle8i Lite.

Answer: You must include the following instruction in your `jserv.properties` file:

```
wrapper.path=C:\orant\bin
```

where `C:\orant\bin` is the directory where (by default) the `OLJDBC40.DLL` lives.

Note that this is *not* `wrapper.classpath`, it's `wrapper.path`.

How Do I Handle Multi-Valued HTML Form Parameters?

Is there any way to handle multi-valued HTML <form> parameters which are needed for <input type="checkbox">?

Answer: There is no built-in way, but you could use a custom Action Handler like this:

```
// MultiValuedParam: XSQL Action Handler that takes the value of
// ----------------  a multi-valued HTTP request parameter and
//                   sets the value of a user-defined page-parameter
//                   equal to the concatenation of the multiple values
//                   with optional control over the separator used
//                   between values and delimiter used around values.
//                   Subsequent actions in the page can then reference
//                   the value of the user-defined page-parameter.
import oracle.xml.xsql.*;
import javax.servlet.http.*;
import org.w3c.dom.*;
public class MultiValuedParam extends XSQLActionHandlerImpl {
  public void handleAction(Node root) {
    XSQLPageRequest req = getPageRequest();
    // Only bother to do this if we're in a Servlet environment
    if (req.getRequestType().equals("Servlet")) {
      Element      actElt = getActionElement();
      // Get name of multi-valued parameter to read from attribute
      String paramName = getAttributeAllowingParam("name",actElt);
      // Get name of page-param to set with resulting value
      String pageParam = getAttributeAllowingParam("page-param",actElt);
      // Get separator string
      String separator = getAttributeAllowingParam("separator",actElt);
      // Get delimiter string
      String delimiter = getAttributeAllowingParam("delimiter",actElt);
      // If the separator is not specified or is blank, use comma
      if (separator == null || separator.equals("")) {
        separator = ",";
      }
      // We're in a Servlet environment, so we can cast
      XSQLServletPageRequest spReq = (XSQLServletPageRequest)req;
      // Get hold of the HTTP Request
      HttpServletRequest httpReq = spReq.getHttpServletRequest();
      // Get the String array of parameter values
      String[] values = httpReq.getParameterValues(paramName);
      StringBuffer str = new StringBuffer();
      // If some values have been returned
      if (values != null) {
```

```
        int items = values.length;
        // Append each value to the string buffer
        for (int z = 0; z < items; z++) {
          // Add a separator before all but the first
          if (z != 0) str.append(separator);
          // Add a delimiter around the value if non-null
          if (delimiter != null) str.append(delimiter);
          str.append(values[z]);
          if (delimiter != null) str.append(delimiter);
        }
        // If page-param attribute not provided, default page param name
        if (pageParam == null) {
          pageParam = paramName+"-values";
        }
        // Set the page-param to the concatenated value
        req.setPageParam(pageParam,str.toString());
      }
    }
  }
}
```

Then you can use this custom action in a page like this:

```
<page xmlns:xsql="urn:oracle-xsql">
  <xsql:action handler="MultiValuedParam" name="guy" page-param="p1" />
  <xsql:action handler="MultiValuedParam" name="guy" page-param="p2"
               delimiter="'" />
  <xsql:action handler="MultiValuedParam" name="guy" page-param="p3"
               delimiter="&quot;" separator=" " />
  <xsql:include-param name="p1"/>
  <xsql:include-param name="p2"/>
  <xsql:include-param name="p3"/>
</page>
```

If this page is requested with the URL following, containing multiple parameters of the same name to produce a multi-valued attribute:

```
http://yourserver.com/page.xsql?guy=Curly&guy=Larry&guy=Moe
```

then the page returned will be:

```
<page>
  <p1>Curly,Larry,Moe</p1>
  <p2>'Curly','Larry','Moe'</p2>
  <p3>"Curly" "Larry" "Moe"</p3>
</page>
```

You can also use the value of the multi-valued page parameter precedingnonzero in a SQL statement by using the following code:

```
<page connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:action handler="MultiValuedParam" name="guy" page-param="list"
               delimiter="'" />
  <!-- Above custom action sets the value of page param named 'list' -->
  <xsql:query>
    SELECT * FROM sometable WHERE name IN ({@list})
  </xsql:query>
</page>
```

Can I Run the XSQL Servlet with Oracle 7.3?

Is there anything that prevents me from running the XSQL Servlet with Oracle 7.3? I know the XML SQL Utility (XSU) can be used with Oracle 7.3 as long as I use it as a client-side utility.

Answer: No. Just make sure you're using the Oracle9*i* JDBC driver, which can connect to an Oracle 7.3 database with no problems.

Why Isn't the Out Variable Supported in <xsql:dml>?

I using `<xsql:dml>` to call a stored procedure which has one `OUT` parameter, but I was not able to see any results. The executed code results in the following statement:

```
<xsql-status action="xsql:dml" rows="0"/>
```

Answer: You cannot set parameter values by binding them in the position of `OUT` variables in this release using `<xsql:dml>`. Only `IN` parameters are supported for binding. You can create a wrapper procedure that constructs XML elements using the HTP package and then your XSQL page can invoke the wrapper procedure using `<xsql:include-owa>` instead.

For an example, suppose you had the following procedure:

```
CREATE OR REPLACE PROCEDURE addmult(arg1        NUMBER,
                                    arg2        NUMBER,
                                    sumval  OUT NUMBER,
                                    prodval OUT NUMBER) IS
BEGIN
  sumval := arg1 + arg2;
  prodval := arg1 * arg2;
```

```
END;
```

You could write the following procedure to wrap it, taking all of the `IN` arguments that the procedure preceding expects, and then encoding the `OUT` values as a little XML datagram that you print to the OWA page buffer:

```
CREATE OR REPLACE PROCEDURE addmultwrapper(arg1 NUMBER, arg2 NUMBER) IS
  sumval  NUMBER;
  prodval NUMBER;
  xml     VARCHAR2(2000);
BEGIN
  -- Call the procedure with OUT values
  addmult(arg1,arg2,sumval,prodval);
  -- Then produce XML that encodes the OUT values
  xml := '<addmult>'||
         '<sum>'||sumval||'</sum>'||
         '<product>'||prodval||'</product>'||
         '</addmult>';
  -- Print the XML result to the OWA page buffer for return
  HTP.P(xml);
END;
```

This way, you can build an XSQL page like this that calls the wrapper procedure:

```
<page connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:include-owa bind-params="arg1 arg2">
    BEGIN addmultwrapper(?,?); END;
  </xsql:include-owa>
</page>
```

This allows a request like the following:

```
http://yourserver.com/addmult.xsql?arg1=30&arg2=45
```

to return an XML datagram that reflects the `OUT` values like this:

```
<page>
  <addmult><sum>75</sum><product>1350</product></addmult>
</page>
```

Why Am I Receiving "Unable to Connect" Errors?

Experimenting with XSQL I'm unable to connect to a database; I get errors like this running the `helloworld.xsql` example:

```
Oracle XSQL Servlet Page Processor 9.0.0.0.0 (Beta)
```

```
XSQL-007: Cannot acquire a database connection to process page.
Connection refused(DESCRIPTION=(TMP=)(VSNNUM=135286784)(ERR=12505)
(ERROR_STACK=(ERROR=(CODE=12505)(EMFI=4))))
```

Does this mean that it has actually found the config file? I have a user with `scott/tiger` setup.

Answer: Yes. If you get this far, it's actually attempting the JDBC connection based on the `<connectiondef>` info for the connection named `demo`, assuming you didn't modify the `helloworld.xsql` demo page.

By default the `XSQLConfig.xml` file comes with the entry for the `demo` connection that looks like this:

```
<connection name="demo">
  <username>scott</username>
  <password>tiger</password>
  <dburl>jdbc:oracle:thin:@localhost:1521:ORCL</dburl>
  <driver>oracle.jdbc.driver.OracleDriver</driver>
</connection>
```

So the error you're getting is likely because of the following reasons:

1. Your database is not on the `localhost` machine.
2. Your database `SID` is not `ORCL.`
3. Your TNS Listener Port is not 1521.

Make sure those values are appropriate for your database and you should have no problems.

Can I Use Other File Extensions Besides *.xsql?

I want users to think they are accessing HTML files or XML files with extensions `.html` and `.xml` respectively, however I'd like to use XSQL to serve the HTML and XML to them. Is it possible to have the XSQL Servlet recognize files with an extension of `.html` or `.xml` in addition to the default `.xsql` extension?

Answer: Sure. There is nothing sacred about the `*.xsql` extension, it is just the default extension used to recognize XSQL pages. You can modify your servlet engine's configuration settings to associate any extension you like with the `oracle.xml.xsql.XSQLServlet` servlet class using the same technique that was used to associate the `*.xsql` extension with it.

How Do I Avoid Errors for Queries Containing XML Reserved Characters?

I have a page like the following:

```
<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
  SELECT id, REPLACE(company,'&','and') company, balance
    FROM vendors
   WHERE outstanding_balance < 3000
</xsql:query>
```

However, when I try to request the page I get the following error:

```
XSQL-005: XSQL page is not well-formed.
XML parse error at line 4, char 16
Expected name instead of '
```

What's wrong?

Answer: The problem is that the ampersand character (&) and the less than sign (<) are reserved characters in XML because:

- The ampersand character (&) starts the sequence of characters that designates an entity reference like ` ` or `<`
- The less than sign (<) starts the sequence of characters that designates an element like `<SomeElement>`

To include a literal ampersand character or less than character you need to either encode each one as a entity reference like this:

```
<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
  SELECT id, REPLACE(company,'&amp;','and') company, balance
    FROM vendors
   WHERE outstanding_balance &lt; 3000
</xsql:query>
```

Alternatively, you can surround an entire block of text with a so-called CDATA section that begins with the sequence <![CDATA[and ends with a corresponding]]> sequence. All text contained in the CDATA section is treated as literal.

```
<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
<![CDATA[
  SELECT id, REPLACE(company,'&','and') company, balance
    FROM vendors
   WHERE outstanding_balance < 3000
]]>
</xsql:query>
```

Why Do I Get "No Posted Document to Process" When I Try to Post XML?

When I try to click a link to an XSQL page that contains an <xsql:insert-request> tag, I see a message in my page "No Posted Document to Process" and no data gets inserted into the database. What's going on?

Answer: When trying to post XML information to an XSQL page for processing, it must be sent by the HTTP POST method. This can be an HTTP POST-ed HTML Form or an XML document sent by HTTP POST. If you try to use HTTP GET instead, there is no posted document, and hence you get this error. Use HTTP POST instead to have the correct behavior.

Can XSQL Support SOAP?

Can an XSQL page be used to implement a SOAP service so that clients over HTTP use it?

Answer: Sure. Your page can access contents of the inbound SOAP message using the `<xsql:set-page-param>` action's **xpath=**"*XpathExpression*" attribute. Alternatively, your customer action handlers can gain direct access to the posted SOAP message body by calling `getPageRequest().getPostedDocument()`. To create the SOAP response body to return to the client, you can either use an XSLT stylesheet or a custom serializer implementation to write out the XML response in an appropriate SOAP-encoded format.

So, while not automatic, it is possible. See the supplied AirportSOAP demo that comes with the XSQL Pages framework for an example of using an XSQL page to implement a SOAP-based Web Service.

How Do I Pass the Connection for XSQL?

I need to be able to pass the connection for XSQL to use in the request. Is this possible?

Answer: Yes. Just reference an XSQL parameter in your page's **connection** attribute, making sure to define an attribute of the same name to serve as the default value for the connection name. For example:

```
<xsql:query conn="testdb" connection="{@conn}" xmlns:xsql="urn:oracle-xsql">
   :
</xsql:query>
```

If you retrieve this page without any parameters, the value of the `conn` parameter will be `testdb`, so the page will use the connection named `testdb` defined in the

XSQLConfig.xml file. If instead you request the page with conn=proddb, then the page will use the connection named proddb instead.

How Do I Control How Database Connections and Passwords Are Stored?

If we need a more sophisticated set of username and password management than the one that is provided by default in XSQL (using the XSQLConfig.xml file) is it possible to override this?

Answer: Yes. You can completely redefine the way the XSQL Page Processor handles database connections by creating your own implementation of the XSQLConnectionManager interface. To achieve this, you need to write a class that implements the oracle.xml.xsql.XSQLConnectionManagerFactory interface and a class that implements the oracle.xml.xsql.XSQLConnectionManager interface, then change the name of the XSQLConnectionManagerFactory class to use in your XSQLConfig.xml configuration file. Once you've done this, your connection management scheme will be used instead of the XSQL Pages default scheme.

How Do I Access Authentication Information in a Custom Connection Manager?

We want to use the HTTP authentication mechanism to get the username and password to connect to the database. Is it possible to get this kind of information in a custom connection manager's getConnection() method?

Answer: Yes. The getConnection() method is passed an instance of the XSQLPageRequest interface. From it, you can get the HTTP Request object by:

1. Testing the request type to make sure it's "Servlet"
2. Casting XSQLPageRequest to XSQLServletPageRequest
3. Calling getHttpServletRequest() on the result of (2)

You can then get the authentication information from that HTTP Request object.

How Do I Retrieve the Name of the Current XSQL Page?

Is there a smart way for an XSQL page to access its own name in a generic way at runtime for the purpose of constructing links to the current page?

Answer: You can use a helper method like this to retrieve the name of the page inside a custom action handler:

```
// Get the name of the current page from the current page's URI
```

```
  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;
  }
```

How Do I Resolve Errors When I Try to Use the FOP Serializer?

I get an error trying to use the FOP Serializer to produce PDF output from my XSQL Page. What could be wrong?

Answer: Typically the problem is that you do not have all of the required JAR files in the CLASSPATH. The XSQLFOPSerializer class lives in the separate `xsqlserializers.jar` file, and this must be in the CLASSPATH to use the FOP integration. Then, the XSQLFOPSerializer class itself has dependencies on several libraries from Apache. For example, here is the source code for a FOP Serializer that works with the Apache FOP 0.20.3RC release candidate of the FOP software:

```
package sample;
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 SampleFOPSerializer 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);
    }
  }
}
```

This FOP serializer depends on having the following additional Apache JAR files in the CLASSPATH at runtime:

1. `fop.jar` - Apache FOP Rendering Engine
2. `batik.jar` - Apache Batik SVG Rendering Engine
3. `avalon-framework-4.0.jar` - API's for Apache Avalon Framework
4. `logkit-1.0.jar` - API's for the Apache Logkit

How Do I Tune XSQL Pages for Fastest Performance?

What recommendations can you provide to make my XSQL pages run the fastest?

Answer: The biggest thing that affects the performance is the size of the data you're querying (and of course the pure speed of the queries). Assuming you have tuned your queries and used true ? bind variables instead of lexical bind variables wherever allowed by SQL, then the key remaining tip is to make sure you are only querying the minimum amount of data needed to render the needed result.

If you are querying thousands of rows of data, only to use your XSLT stylesheet to filter the rows to present only 10 of those rows in the browser, then this is a bad choice. Use the database's capabilities to the maximum to filter the rows and return only the 10 rows you care about if at all possible. Think of XSQL as a thin

coordination layer between Oracle database and the power of XSLT as a transformation language.

How Do I Use XSQL with Other Connection Pool Implementations?

Can you set up XSQL pages to use connections taken from a connection pool? For example, if you are running XSQL servlet in a Weblogic web server, how would the connection definition have to be set up to take a connection from the existing pool?

Answer: XSQL implements it's own connection pooling so in general you don't have to use another connection pool, but if providing the JDBC connection string of appropriate format is not enough to use the WebLogic pool, then you can create your own custom connection manager for XSQL by implementing the interfaces `XSQLConnectionManagerFactory` and `XSQLConnectionManager`.

How Do I Include XML Documents Stored in CLOBs?

How do I include XML documents stored in a CLOB in the database into my XSQL page?

Answer: Use `<xsql:include-xml>` with a query to retrieve the CLOB value.

How Do I Combine JSP and XSQL in the Same Page?

Is it possible to combine XSQL and JSP tags in the same page or should one use include tags for that?

Answer: JSP and XSQL are two different models. JSP is a model that is based on writing streams of characters to an output stream. XSQL is a model that is pure XML / XSLT-based. At the end of the day, some result like HTML or XML comes back to the user, and there really isn't anything that you can implement with XSQL that you could not implement in JSP by writing code and working with XML documents as streams of characters, doing lots of internal reparsing. XSQL fits the architecture when customers want to cleanly separate the data content (represented in XML) from the data presentation (represented by XSLT stylesheets). Since it specializes in this XML / XSLT architecture, it is optimized for doing that.

You can, for example, use `<jsp:include>` or `<jsp:forward>` to have a JSP page include / forward to an XSQL page. This is the best approach.

Can I Choose a Stylesheet Based on Input Arguments?

Is it possible to change stylesheets dynamically based on input arguments?

Answer: Sure. Yes, you can achieve this by using a lexical parameter in the href attribute of your xml-stylesheet processing instruction.

```
<?xml-stylesheet type="text/xsl" href="{@filename}.xsl"?>
```

The value of the parameter can be passed in as part of the request, or by using the `<xsql:set-page-param>` you can set the value of the parameter based on a SQL query.

10

XDK JavaBeans

This chapter describes the JavaBeans available for use with Oracle XML. The topics in this chapter are:

Accessing Oracle XML Transviewer Beans

The Oracle XML Transviewer beans are provided as part of XDK for JavaBeans with the Oracle9*i* Enterprise and Standard Editions from Release 2 (8.1.6) and higher. If you do not have these editions, then you can download the beans from the site: http://otn.oracle.com/tech/xml

XDK for Java: XML Transviewer Bean Features

XML Transviewer Beans facilitate the addition of graphical interfaces to your XML applications.

Direct Access from JDeveloper

Bean encapsulation includes documentation and descriptors that can be accessed directly from Java Integrated Development Environments like JDeveloper.

Sample Transviewer Bean Application

A sample application that demonstrates all of the beans to create a simple XML editor and XSL transformer is included with your software.

The included sample application with the XML SQL Utility (XSU) cause the following:

- Database queries to materialize XML
- Transform the XML through an XSL stylesheet
- Store the resulting XML document in the database for fast retrieval

Database Connectivity

Database Connectivity is included with the XML Transviewer Beans. The beans can now connect directly to a JDBC-enabled database to retrieve and store XML and XSL files.

XML Transviewer Beans

XML Transviewer Beans comprises the following beans:

DOMBuilder Bean

The DOMBuilder bean is a non-visual bean. It builds a DOMTree from an XML document.

The DOMBuilder bean encapsulates the XML Parser for Java's DOMParser class with a bean interface and extends its functionality to permit asynchronous parsing. By registering a listener, Java applications can parse large or successive documents and then allow control to return immediately to the caller.

See Also: "Using DOMBuilder Bean" on page 10-5

XSLTransformer Bean

The XSLTransformer bean is a non-visual bean. It accepts an XML file, applies the transformation specified by an input XSL stylesheet and creates the resulting output file.

XSLTransformer bean enables you to transform an XML document to almost any text-based format including XML, HTML, and DDL, by applying the appropriate XSL stylesheet.

- When integrated with other beans, XSLTransformer bean enables an application or user to view the results of transformations immediately.
- This bean can also be used as the basis of a server-side application or servlet to render an XML document, such as an XML representation of a query result, into HTML for display in a browser.

See Also: "Using XSLTransformer Bean" on page 10-9

Treeviewer Bean

The Treeviewer bean displays XML formatted files graphically as a tree. The branches and leaves of this tree can be manipulated with a mouse.

See Also: "Using Treeviewer Bean" on page 10-13

XMLSourceView Bean

The XMLSourceView bean is a visual Java bean. It allows visualization of XML documents and editing. It enables the display of XML and XSL formatted files with color syntax highlighting when modifying an XML document with a text editor. This helps view and edit the files. It can be integrated with DOMBuilder bean, and allows pre- or post-parsing visualization and validation against a specified DTD.

See Also: "Using XMLSourceView Bean" on page 10-15

XMLTransformPanel Bean

This is a visual Java bean that applies XSL transformations on XML documents and shows the results. It allows editing of XML and XSL input files.

See Also: "Using XMLTransformPanel Bean" on page 10-20

DBViewer Bean

DBViewer bean is Java bean that displays database queries or any XML by applying XSL stylesheets and visualizing the resulting HTML in a scrollable swing panel. DBViewer bean has XML and XSL tree buffers as well as a result buffer. DBViewer bean allows the calling program to:

- Load or save buffers from various sources such as from CLOB tables in an Oracle database or from the file system. Control can be also used to move files between the file system and the user schema in the database.
- Apply stylesheet transformations to the XML buffer using the stylesheet in the XSL buffer.

The result can be stored in the result buffer. The *XML* and *XSL* buffer content can be shown as a source or tree structure. The *result* buffer content can be rendered as HTML and also shown as source or tree structure. The XML buffer can be loaded from a database query.

DBAccess Bean

DBAccess bean maintains CLOB tables that contain multiple XML and text documents.

Using the XML Transviewer Beans

The guidelines for using the XML Transviewer Beans are described in the following sections:

- Using DOMBuilder Bean
- Using XSLTransformer Bean
- Using Treeviewer Bean
- Using XMLSourceView Bean

- Using XMLTransformPanel Bean
- Using DBViewer Bean
- Using DBAccess Bean
- Using the XMLDiff Bean

See Also:

- *Oracle9i XML API Reference - XDK and Oracle XML DB*

Using DOMBuilder Bean

DOMBuilder() class 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 and the DOMBuilderListener interface must be used for notification when the tree is built.

Used for Asynchronous Parsing in the Background

The DOMBuilder bean encapsulates the XML Parser for Java with a bean interface. It extends its functionality to permit asynchronous parsing. By registering a listener, a Java application can parse documents and return control return to the caller.

Asynchronous parsing in a background thread can be used interactively in visual applications. For example, when parsing a large file with the normal parser, the user interface freezes until the parsing has completed. This can be avoided with the DOMBuilder bean. After calling the DOMBuilder bean parse method, the application can immediately regain control and display “Parsing, please wait”. If a “Cancel” button is included you can also cancel the operation. The application can continue when `domBuilderOver()` method is called by DOMBuilder bean when background parsing task has completed.

DOMBuilder Bean Parses Many Files Fast

When parsing a large number of files, DOMBuilder bean can save time. Response times that are up to 40% faster have been recorded when compared to parsing the files one by one.

DOMBuilder Bean Usage

Figure 10–1 illustrates DOMBuilder Bean usage.

1. The XML document to be parsed is input as a file, string buffer, or URL.
2. This inputs the method `DOMBuilder.addDOMBuilderListener(DOMBuilderListener)` and adds DOMBuilderListener.
3. The DOMBuilder.parser() method parses the XML document.
4. Optionally, the parsed result undergoes further processing.

   See Also : Table 10–1 for a list of available methods to apply

5. DOMBuilderListener API is called using `DOMBuilderOver()` method. This is called when it receives an asynchronous call from an application. This interface must be implemented to receive notifications about events during asynchronous parsing. The class implementing this interface must be added to the DOMBuilder using `addDOMBuilderListener` method.

   Available DOMBuilderListener methods are:

   - `domBuilderError(DOMBuilderEvent)`. This method is called when parse errors occur.
   - `domBuilderOver(DOMBuilderEvent)`. This method is called when the parse completes.
   - `domBuilderStarted(DOMBuilderEvent)`. This method is called when parsing begins.

6. `DOMBuilder.getDocument()` fetches the resulting DOM document and outputs the DOM document.

Figure 10–1 DOMBuilder Bean Usage

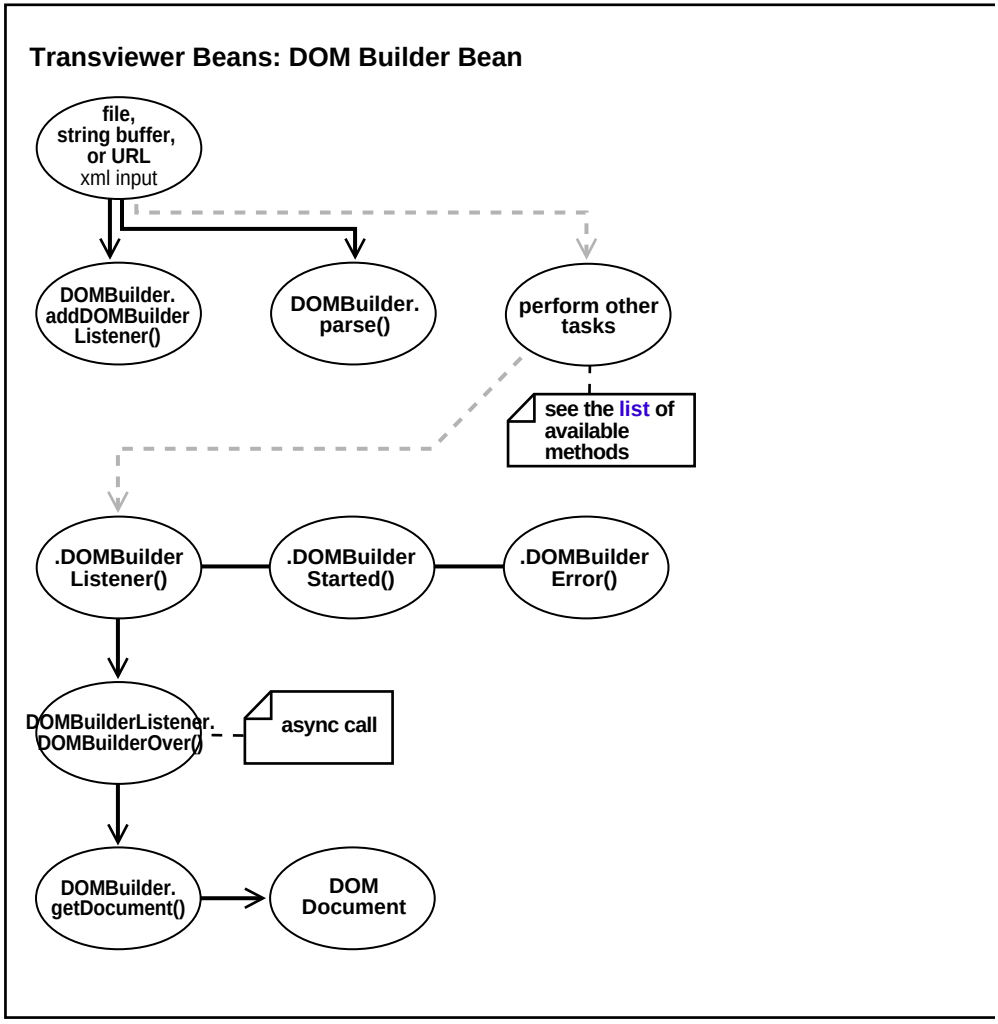


Table 10–1 DOMBuilder Bean: Methods

Method	Description
addDOMBuilderErrorListener(DOMBuilderErrorListener)	Adds DOMBuilderErrorListener.
addDOMBuilderListener(DOMBuilderListener)	Adds DOMBuilderListener.
	Get the DTD.
getDocument()	Gets the document.
getId()	Returns the parser object id.
getReleaseVersion()	Returns the release version of the Oracle XML Parser.
	Gets the document.
getValidationMode()	Returns the validation mode.
parse(InputSource)	Parses the XML from given input source.
	Parses the XML from given input stream.
parse(Reader)	Parses the XML from given input stream.
parse(String)	Parses the XML from the URL indicated.
parse(URL)	Parses the XML document pointed to by the given URL and creates the corresponding XML document hierarchy.
parseDTD(InputSource, String)	Parses the XML External DTD from given input source.
parseDTD(InputStream, String)	Parses the XML External DTD from given input stream.
parseDTD(Reader, String)	Parses the XML External DTD from given input stream.
	Parses the XML External DTD from the URL indicated.
parseDTD(URL, String)	Parses the XML External DTD document pointed to by the given URL and creates the corresponding XML document hierarchy.
removeDOMBuilderErrorListener(DOMBuilderErrorListener)	Removes DOMBuilderErrorListener.
removeDOMBuilderListener(DOMBuilderListener)	Removes DOMBuilderListener.
run()	This method runs in a thread.

Table 10–1 DOMBuilder Bean: Methods (Cont.)

Method	Description
	Set the base URL for loading external enitites and DTDs.
setDebugMode(boolean)	Sets a flag to turn on debug information in the document.
setDoctype(DTD)	Sets the DTD.
setErrorStream(OutputStream)	Creates an output stream for the output of errors and warnings.
setErrorStream(OutputStream, String)	Creates an output stream for the output of errors and warnings.
setErrorStream(PrintWriter)	Creates an output stream for the output of errors and warnings.
	Sets the node factory.
setPreserveWhitespace(boolean)	Sets the white space preserving mode.
setValidationMode(boolean)	Sets the validation mode.
showWarnings(boolean)	Switches to determine whether to print warnings.

Using XSLTransformer Bean

The XSLTransformer bean accepts an XML file and applies the transformation specified by an input XSL stylesheet to create and output file. It enables you to transform an XML document to almost any text-based format including XML, HTML, and DDL, by applying an XSL stylesheet.

When integrated with other beans, XSLTransformer bean enables an application or user to immediately view the results of transformations.

This bean can also be used as the basis of a server-side application or servlet to render an XML document, such as an XML representation of a query result, into HTML for display in a browser.

The XSLTransformer bean encapsulates the Java XML Parser XSLT processing engine with a bean interface and extends its functionality to permit asynchronous transformation.

By registering a listener, your Java application can transform large and successive documents by having the control returned immediately to the caller.

Do You Have Many Files to Transform? Use XSLTransformer Bean

XSL transformations can be time consuming. Use XSL Transformer bean in applications that transform large numbers of files and it can concurrently transform multiple files.

Do You Need a Responsive User Interface? Use XSLTransformer Bean

XSLTransformer bean can be used for visual applications for a responsive user interface. There are similar issues here as with DOMBuilder bean.

By implementing `XSLTransformerListener()` method, the caller application can be notified when the transformation is complete. The application is free to perform other tasks in between requesting and receiving the transformation.

XSL Transviewer Bean Scenario 1: Regenerating HTML Only When Data Changes

This scenario illustrates one way of applying XSLTransformer bean.

1. Create a SQL query. Store the selected XML data in a CLOB table.
2. Using the XSLTransfomer bean, create an XSL stylesheet and interactively apply this to the XML data until you are satisfied with the data presentation. This can be HTML produced by the XSL transformation.
3. Now that you have the desired SQL (data selection) and XSL (data presentation), create a trigger on the table or view used by your SQL query. The trigger can execute a stored procedure. The stored procedure, can for example, do the following:
   - Run the query
   - Apply the stylesheet
   - Store the resulting HTML in a CLOB table
4. This process can repeat whenever the source data table is updated.

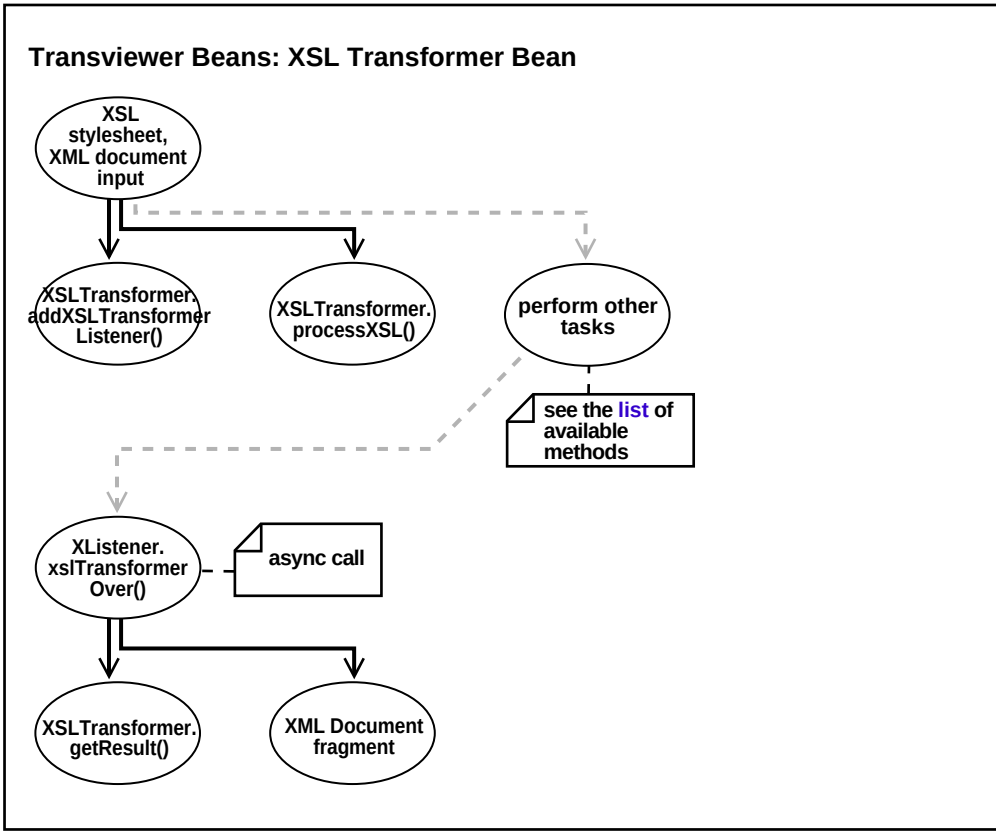
   The HTML stored in the CLOB table always mirrors the last data stored in the tables being queried. A JSP (Java Server Page) can display the HTML.

   In this scenario, multiple end users do not produce multiple data queries that contribute to larger loads to the database. The HTML is regenerated only when the underlying data changes.

XSLTransformer Bean Usage

Figure 10–2 illustrates XSLTransformer bean usage. For examples of implementing this bean, see "Transviewer Bean Example 1: AsyncTransformSample.java".

Figure 10–2 XSLTransformer Bean Usage



1. An XSL stylesheet and XML document input the XSLTransformer using the `XSLTransfomer.addXSLTransformerListener(XSLTransformerListener)` method. This adds a listener.
2. The XSLTransfomer.processXSL() method initiates the XSL transformation in the background.

3. Optionally, other work can be assigned to the XSLTransformer bean. Table 10–2 lists the XSLTransformer bean methods.
4. When the transformation is complete, an asynchronous call is made and the `XSLTransformerListener.xslTransformerOver()` method is called. This interface must be implemented to receive notifications about events during the asynchronous transformation. The class implementing this interface must be added to the XSLTransformer event queue using the method `addXSLTransformerListener`.
5. The XSLTransformer.getResult() method returns the XML document fragment for the resulting document.
6. It ouputs the XML document fragment.

Table 10–2 XSLTransformer Bean: Methods

Method	Description
addXSLTransformerErrorListener(XSLTransformerErrorListener)	Adds an error event listener.
addXSLTransformerListener(XSLTransformerListener)	Adds a listener.
getId()	Returns the unique XSLTransformer id.
getResult()	Returns the document fragment for the resulting document.
processXSL(XSLstylesheet, InputStream, URL)	Initiates XSL Transformation in the background.
processXSL(XSLstylesheet, Reader, URL)	Initiates XSL Transformation in the background.
processXSL(XSLstylesheet, URL, URL)	Initiates XSL Transformation in the background.
processXSL(XSLstylesheet, XMLDocument)	Initiates XSL Transformation in the background.
processXSL(XSLstylesheet, XMLDocument, OutputStream)	Initiates XSL Transformation in the background.
removeDOMTransformerErrorListener(XSLTransformerErrorListener)	Removes an error event listener.
removeXSLTransformerListener(XSLTransformerListener)	Removes a listener.
run()	
setErrorStream(OutputStream)	Sets the error stream used by the XSL processor.
showWarnings(boolean)	Sets the showWarnings flag used by the XSL processor.

Using Treeviewer Bean

The Treeviewer bean displays an XML document as a tree. It recognizes the following XML DOM nodes:

- Tag
- Attribute Name
- Attribute Value
- Comment
- CDATA
- PCDATA
- PI Data
- PI Name
- NOTATION Symbol

It takes as input an org.w3c.dom.Document object.

Figure 10–3, "Treeviewer Bean in Action: Displaying an XML Document as a Tree" shows how the Treeviewer bean displays the XML document and the editing options.

Figure 10–3 Treeviewer Bean in Action: Displaying an XML Document as a Tree

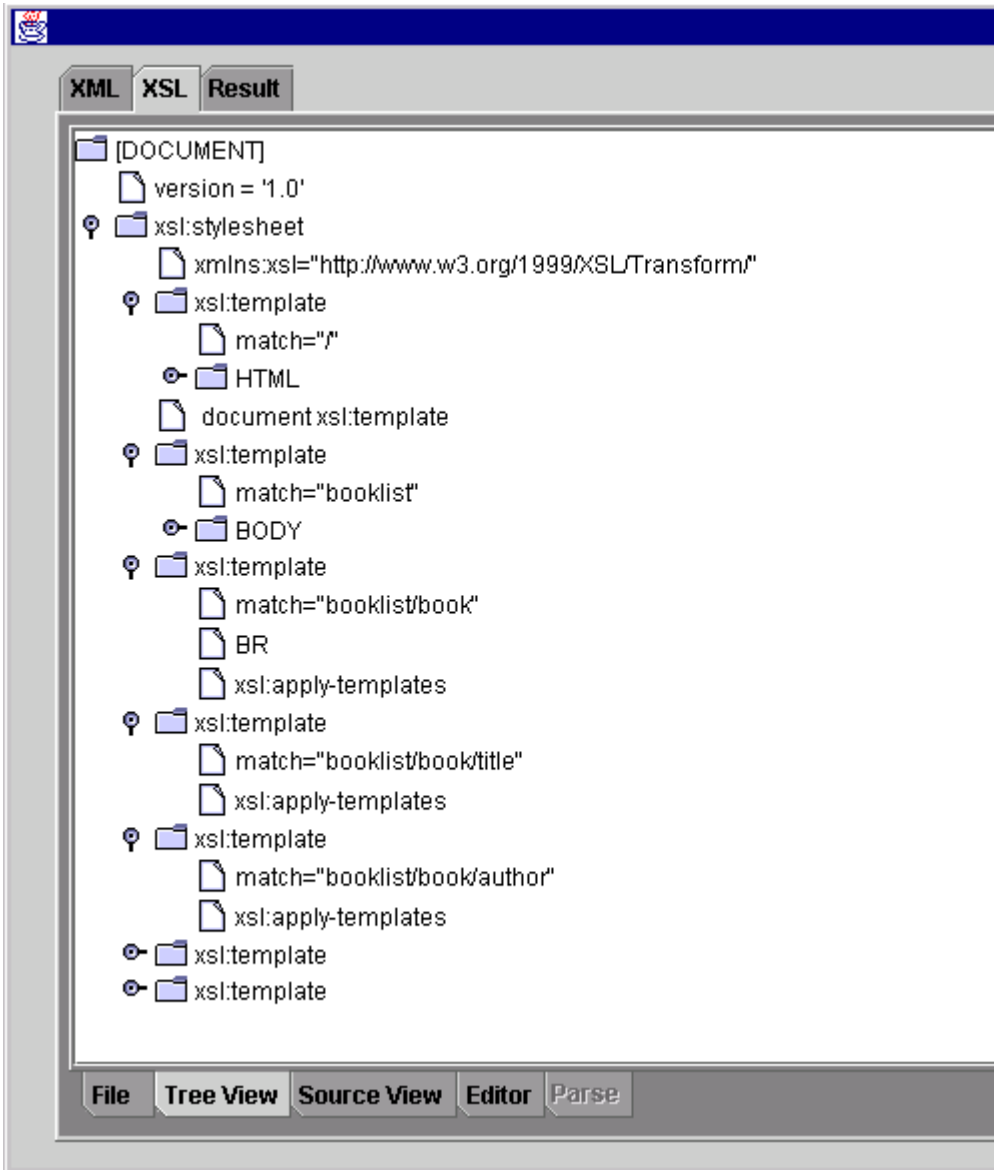
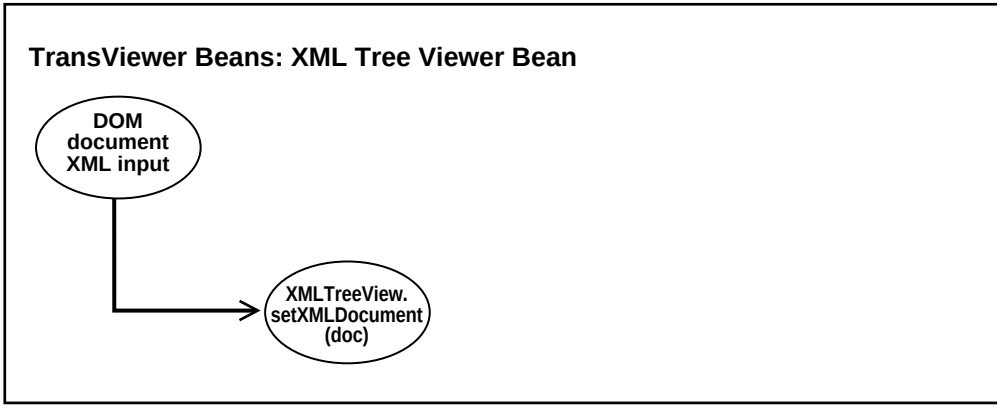


Figure 10–4 illustrates XML Treeviewer bean usage. A DOM XML document is input to the `XMLTreeView.setXMLDocument(doc)` method. This associates the XML Treeviewer with the XML document. The Treeviewer bean methods are:

- getPreferredSize()—Returns the XMLTreeView preferred size
- setXMLDocument(Document)—Associates the XMLTreeViewer with an XML document
- updateUI()—Forces the XMLTreeView to update / refresh the user interface

Figure 10–4 XML Treeviewer Bean Usage



Using XMLSourceView Bean

XMLSourceView bean is a visual Java bean that displays an XML document. It improves the viewing of XML and XSL files by color-highlighting the XML / XSL syntax. It also offers an Edit mode. XMLSourceView bean easily integrates with DOMBuilder bean. It allows for pre- or post-parsing visualization and validation against a specified DTD.

XMLSourceView bean recognizes the following XML token types:

- Tag
- Attribute Name
- Attribute Value
- Comment

- CDATA
- PCDATA
- PI Data
- PI Name
- NOTATION Symbol

Each token type has a foreground color and font. The default color / font settings can be changed by the user. This takes an org.w3c.dom.Document object as input.

XMLSourceView Bean Usage

Figure 10–5 displays an XML document with tags shown in blue, tag content in black, and attributes in red.

Figure 10–6 shows the XMLSourceView bean usage. This is part of the `oracle.xml.srcviewer` API. A DOM document inputs XMLSourceView.SetXMLDocument(Doc). The resulting DOM document is displayed. See "Transviewer Bean Example 2: ViewSample.java".

Figure 10–5 XMLSourceView Bean in Action: Displaying an XML Document with Color Highlighting

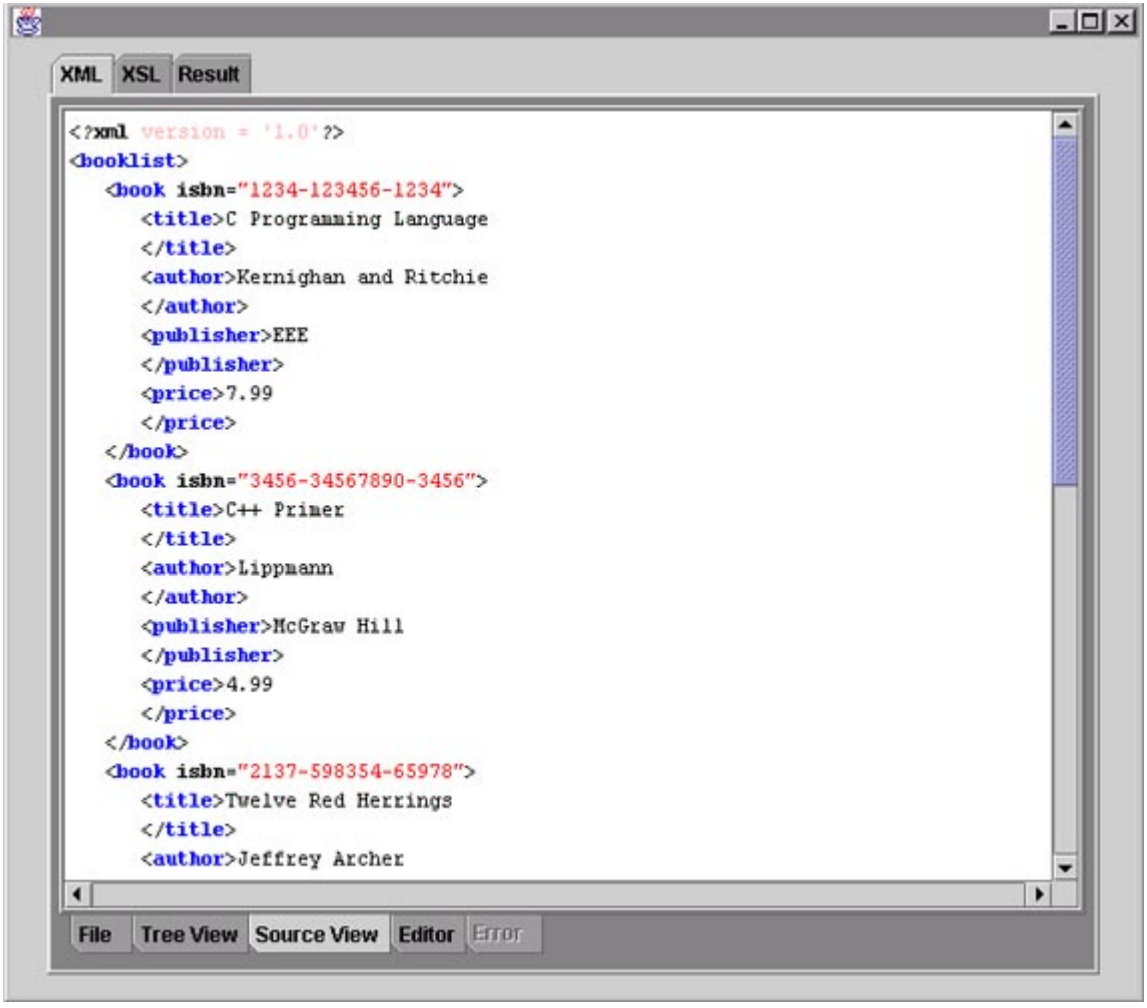
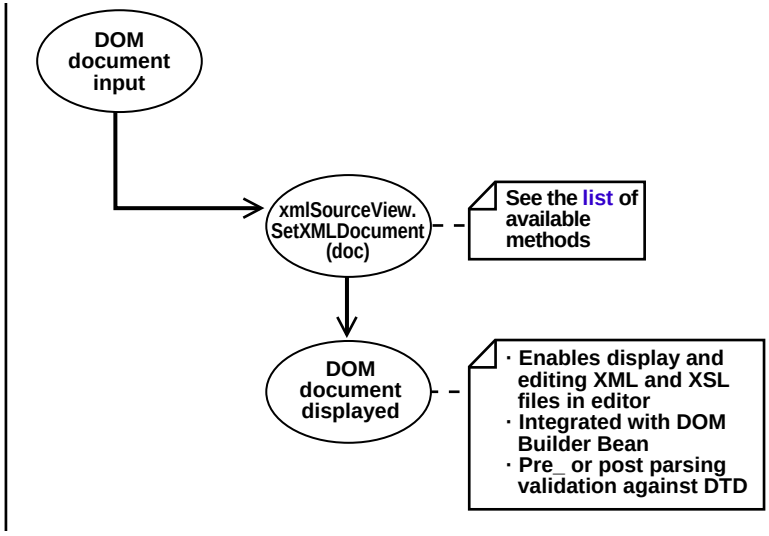


Figure 10–6 XMLSourceView Bean Usage



The following table, Table 10–3, lists the XMLSourceView Bean methods.

Table 10–3 XMLSourceView Bean Methods

Method	Description
fontGet(AttributeSet)	Extracts and returns the font from a given attributeset.
fontSet(MutableAttributeSet, Font)	Sets the mutableattributeset font.
getAttributeNameFont()	Returns the Attribute Value font.
getAttributeNameForeground()	Returns the Attribute Name foreground color.
getAttributeValueFont()	Returns the Attribute Value font.
getAttributeValueForeground()	Returns the Attribute Value foreground color.
getBackground()	Returns the background color.
getCDATAFont()	Returns the CDATA font.
getCDATAForeground()	Returns the CDATA foreground color.
getCommentDataFont()	Returns the Comment Data font.
getCommentDataForeground()	Returns the Comment Data foreground color.
getEditedText()	Returns the edited text.

Table 10–3 XMLSourceView Bean Methods (Cont.)

Method	Description
getJTextPane()	Returns the viewer JTextPane component.
getMinimumSize()	Returns the XMLSourceView minimal size.
getNodeAtOffset(int)	Returns the XML node at a given offset.
getPCDATAFont()	Returns the PCDATA font.
getPCDATAForeground()	Returns the PCDATA foreground color.
getPIDataFont()	Returns the PI Data font.
getPIDataForeground()	Returns the PI Data foreground color.
getPINameFont()	Returns the PI Name font.
getPINameForeground()	Returns the PI Data foreground color.
getSymbolFont()	Returns the NOTATION Symbol font.
getSymbolForeground()	Returns the NOTATION Symbol foreground color.
getTagFont()	Returns the Tag font.
getTagForeground()	Returns the Tag foreground color.
getText()	Returns the XML document as a String.
isEditable()	Returns boolean to indicate whether this object is editable.
selectNodeAt(int)	Moves the cursor to XML Node at offset i.
setAttributeNameFont(Font)	Sets the Attribute Name font.
setAttributeNameForeground(Color)	Sets the Attribute Name foreground color.
setAttributeValueFont(Font)	Sets the Attribute Value font.
setAttributeValueForeground(Color)	Sets the Attribute Value foreground color.
setBackground(Color)	Sets the background color.
setCDATAFont(Font)	Sets the CDATA font.
setCDATAForeground(Color)	Sets the CDATA foreground color.
setCommentDataFont(Font)	Sets the Comment font.
setCommentDataForeground(Color)	Sets the Comment foreground color.
setEditable(boolean)	Sets the specified boolean to indicate whether this object should be editable.

Table 10–3 XMLSourceView Bean Methods (Cont.)

Method	Description
setPCDATAFont(Font)	Sets the PCDATA font.
setPCDATAForeground(Color)	Sets the PCDATA foreground color.
setPIDataFont(Font)	Sets the PI Data font.
setPIDataForeground(Color)	Sets the PI Data foreground color.
setPINameFont(Font)	Sets the PI Name font.
setPINameForeground(Color)	Sets the PI Name foreground color.
setSelectedNode(Node)	Sets the cursor position at the selected XML node.
setSymbolFont(Font)	Sets the NOTATION Symbol font.
setSymbolForeground(Color)	Sets the NOTATION Symbol foreground color.
setTagFont(Font)	Sets the Tag font.
setTagForeground(Color)	Sets the Tag foreground color.
setXMLDocument(Document)	Associates the XMLviewer with a XML document.

Using XMLTransformPanel Bean

XMLTransformPanel visual bean applies XSL transformations to XML documents. It visualizes the result and allows editing of input XML and XSL documents and files. XMLTransformPanel bean requires no programmatic input. It is a component that interacts directly with you and is not customizable.

XMLTransformPanel Bean Features

XMLTransformPanel bean has the following features:

- Imports and exports XML and XSL files from the file system, and XML, XSL, and HTML files from Oracle9*i*. With Oracle9*i*, XMLTransformPanel bean uses two-column CLOB tables. The first column stores the data name (file name) and the second stores the data text (file's data) in a CLOB. The bean lists all CLOB tables in your schema. When you click on a table, the bean lists its file names. You can also create or delete tables and retrieve or add files to the tables. This can be useful for organizing your information. See Figure 10–7.

> **Note:** CLOB tables created by the XSL Transformer bean can be used by trigger-based stored procedures to mirror tables or views in the database *into HTML data* held in these CLOB tables. See "XSL Transviewer Bean Scenario 1: Regenerating HTML Only When Data Changes".

- Supports multiple database connections.
- Creates XML from database result sets. This feature enables you to submit any SQL query to the database that you are currently connected. The bean converts the result set into XML and automatically loads this XML data into the bean's XML buffer for further processing.
- Edits XML and XSL data loaded into the bean.
- Applies XSL transformations to XML buffers and show the results. See With the bean, you can also export results to the file system or a CLOB in the database.

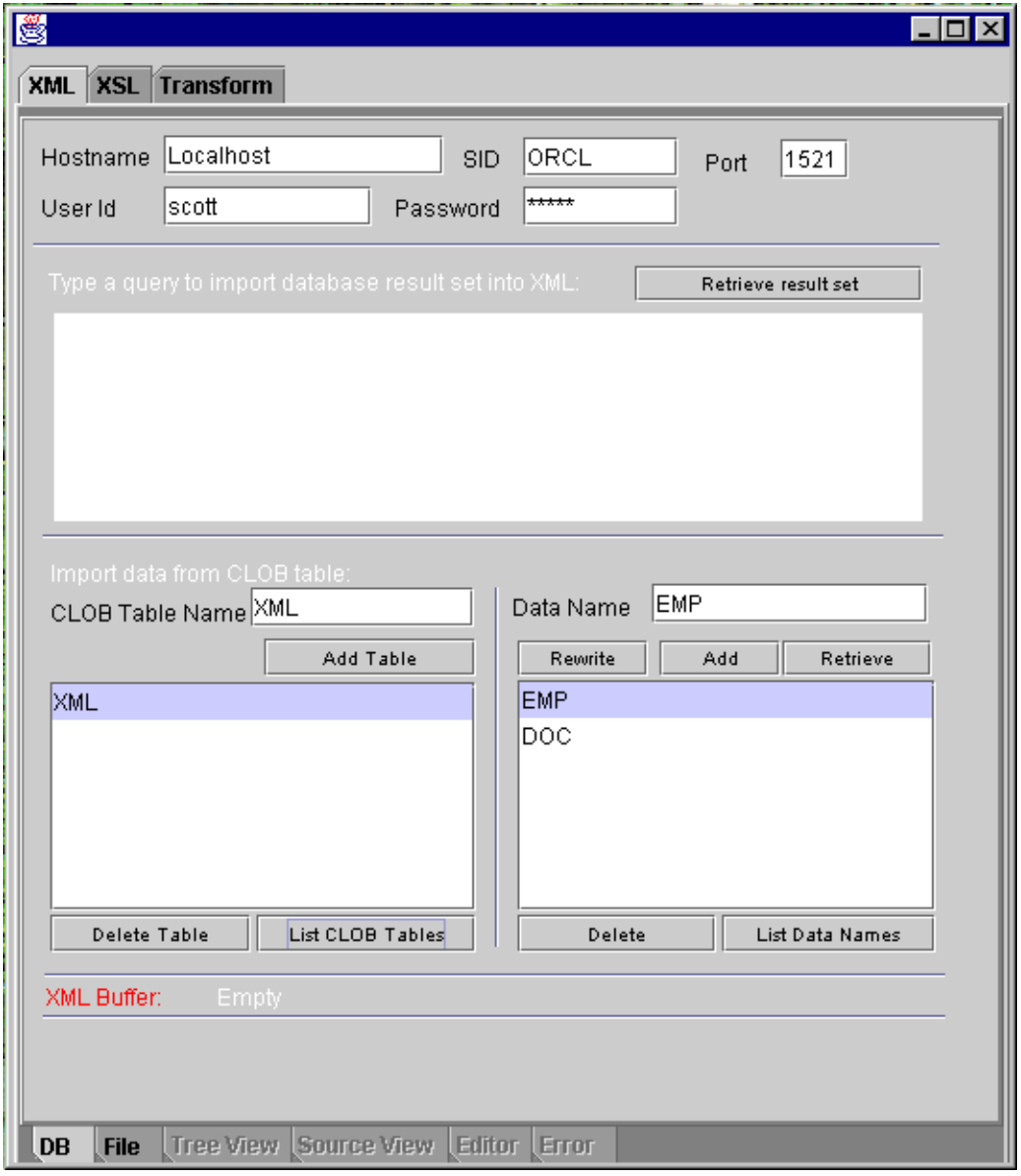
Transviewer Bean Application

The Transviewer bean is one application that illustrates the use of XMLTransform Panel bean. It can be used from a command line to perform the following actions:

- Edit and parse XML files
- Edit and apply XSL transformations
- Retrieve and save XML, XSL and result files in the file system or in Oracle9*i*

> **See Also:** "Transviewer Bean Example 3: XMLTransformPanelSample.java" for an example of how to use XMLTransformPanel.

Figure 10–7 XSLTransformPanel Bean in Action: Showing CLOB Table and Data Names



Using DBViewer Bean

DBViewer bean can be used to display database queries on any XML document by applying XSL stylesheets and visualizing the resulting HTML in a scrollable swing panel. See:

- Figure 10–8, "DBViewer Bean in Action: Entering a Database Query to Generate XML"
- Figure 10–9, "DBViewer Bean in Action: Viewing the XML Document After Transforming to HTML With XSL Style Sheet"

DBViewer bean has the following three buffers:

- XML
- XSL
- Result buffer

DBViewer bean API allows the calling program to load or save buffers from various sources and apply stylesheet transformation to the XML buffer using the stylesheet in the XSL buffer. Results can be stored in the result buffer.

Showing Content

Content in the XML and XSL buffers can be shown as a source or tree structure. Content in the result buffer can be rendered as HTML and also shown as a source or tree structure.

Loading and Saving the Buffers

The XML buffer can be loaded using a database query. All the buffers can be loaded from and files saved from the following:

- CLOB tables in Oracle9*i*
- File system

Therefore, control can also be used to move files between the file system and the user schema in the database.

Figure 10–8 DBViewer Bean in Action: Entering a Database Query to Generate XML

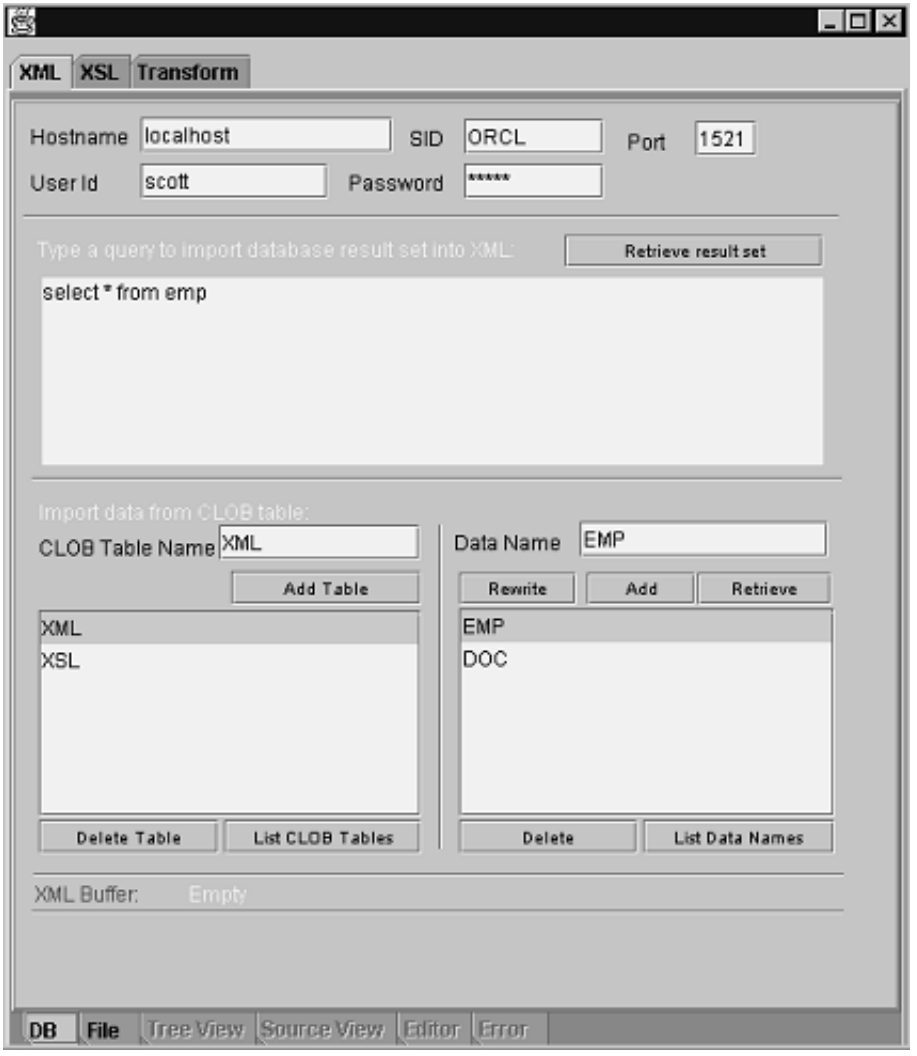
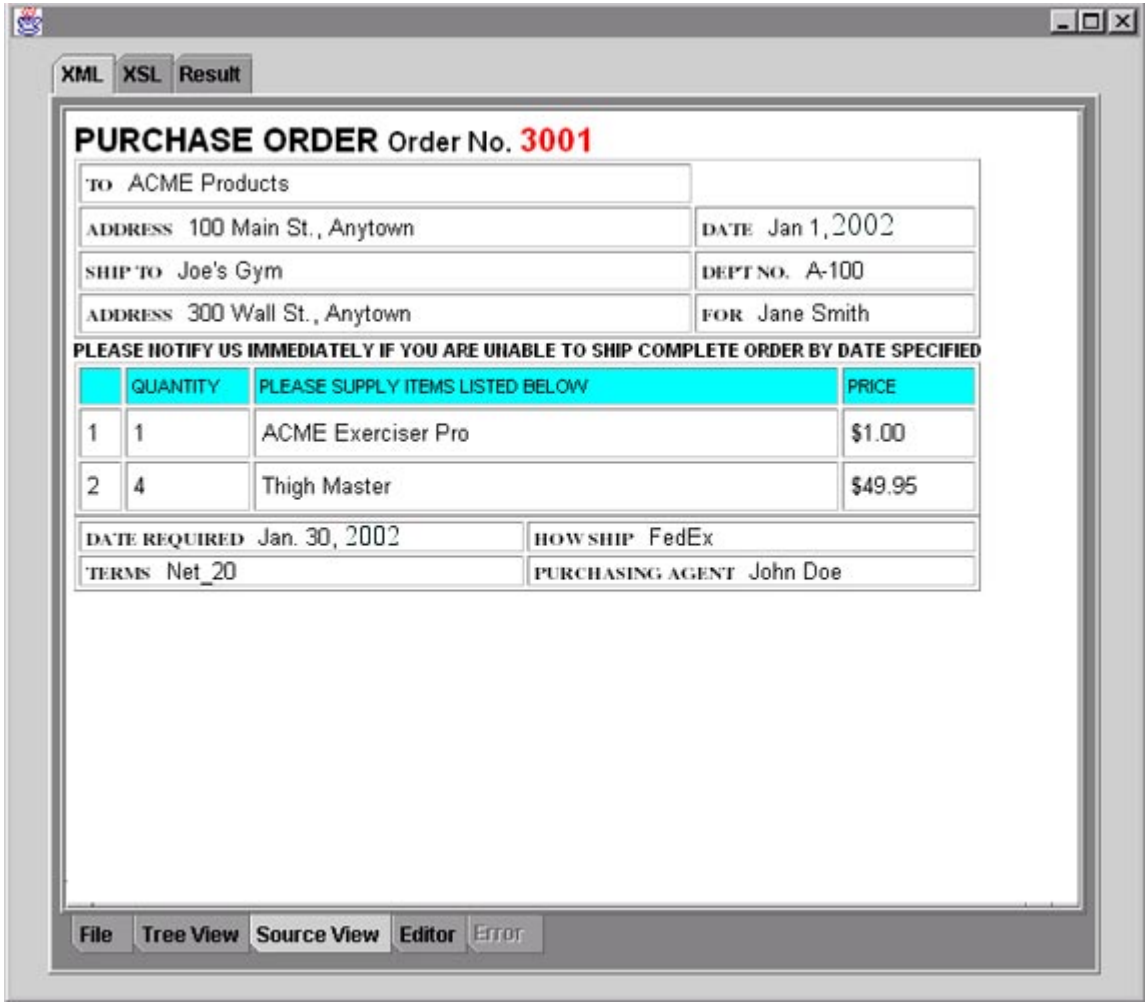


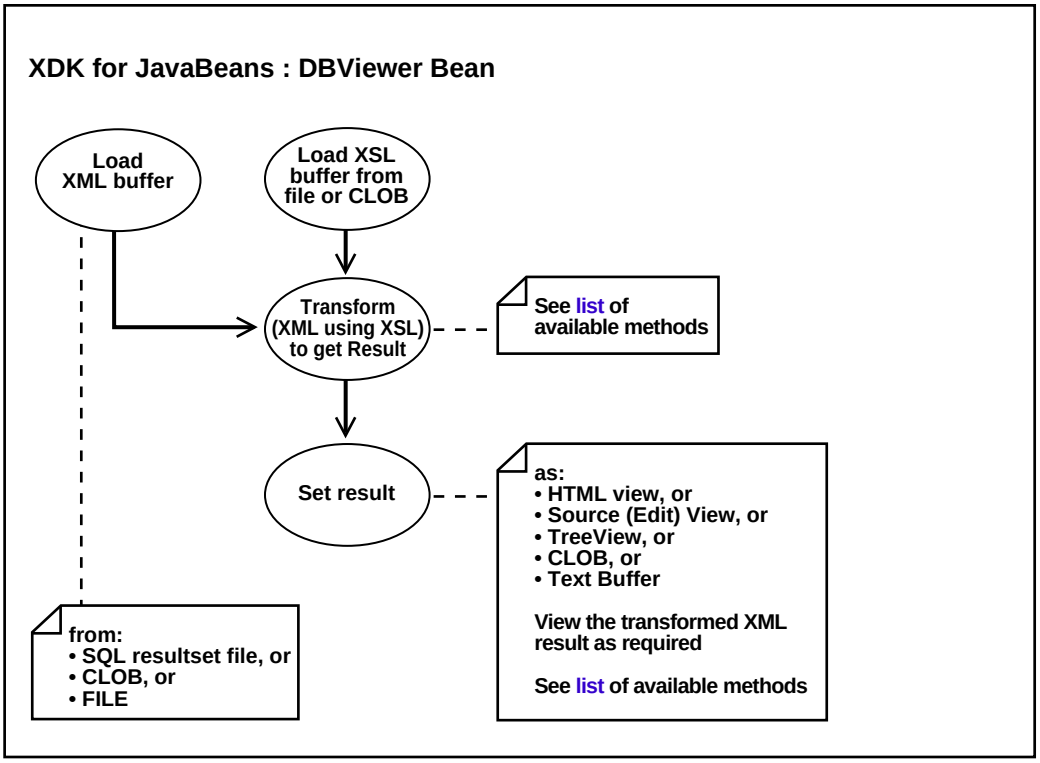
Figure 10–9 DBViewer Bean in Action: Viewing the XML Document After Transforming to HTML With XSL Style Sheet



DBViewer Bean Usage

Figure 10–10 illustrates DBViewer bean's usage.

Figure 10–10 DBViewer Bean Usage Diagram



DBViewer Bean Methods

Table 10–4 lists the DBViewer bean methods.

Table 10–4 DBViewer Bean Methods

Method	Description
DBViewer()	Constructs a new instance.
getHostname()	Gets database host name

Table 10–4 DBViewer Bean Methods (Cont.)

Method	Description
getInstancename()	Gets database instance name.
getPassword()	Gets user password.
getPort()	Gets database port number.
getResBuffer()	Gets the content of the result buffer.
getResCLOBFileName()	Gets result CLOB file name.
getResCLOBTableName()	Gets result CLOB table name.
getResFileName()	Gets Result file name.
getUsername()	Gets user name.
getXmlBuffer()	Gets the content of the XML buffer.
getXmlCLOBFileName()	Gets XML CLOB file name.
getXmlCLOBTableName()	Gets XML CLOB table name.
getXmlFileName()	Gets XML file name.
getXMLStringFromSQL(String)	Gets XML presentation of result set from SQL query.
getXslBuffer()	Gets the content of the XSL buffer.
getXslCLOBFileName()	Gets the XSL CLOB file name.
getXslCLOBTableName()	Gets XSL CLOB table name.
getXslFileName()	Gets XSL file name.
loadResBuffer(String)	Loads the result buffer from file.
loadResBuffer(String, String)	Loads the result buffer from CLOB file.
loadResBufferFromClob()	Loads the result buffer from CLOB file.
loadResBufferFromFile()	Loads the result buffer from file.
loadXmlBuffer(String)	Loads the XML buffer from file.
loadXmlBuffer(String, String)	Loads the XML buffer from CLOB file.
loadXmlBufferFromClob()	Loads the XML buffer from CLOB file.
loadXmlBufferFromFile()	Loads the XML buffer from file.
loadXMLBufferFromSQL(String)	Loads the XML buffer from SQL result set.
loadXslBuffer(String)	Loads the XSL buffer from file.

Table 10–4 DBViewer Bean Methods (Cont.)

Method	Description
loadXslBuffer(String, String)	Loads the XSL buffer from CLOB file.
loadXslBufferFromClob()	Loads the XSL buffer from CLOB file.
loadXslBufferFromFile()	Loads the XSL buffer from file.
parseResBuffer()	Parses the result buffer and refresh the tree view and source view.
parseXmlBuffer()	Parses the XML buffer and refresh the tree view and source view.
parseXslBuffer()	Parses the XSL buffer and refresh the tree view and source view.
saveResBuffer(String)	Saves the result buffer to file.
saveResBuffer(String, String)	Saves the result buffer to CLOB file.
saveResBufferToClob()	Saves the result buffer to CLOB file.
saveResBufferToFile()	Saves the result buffer to file.
saveXmlBuffer(String)	Saves the XML buffer to file.
saveXmlBuffer(String, String)	Saves the XML buffer to CLOB file.
saveXmlBufferToClob()	Saves the XML buffer to CLOB file.
saveXmlBufferToFile()	Saves the XML buffer to file.
saveXslBuffer(String)	Saves the XSL buffer to file.
saveXslBuffer(String, String)	Saves the XSL buffer to CLOB file.
saveXslBufferToClob()	Saves the XSL buffer to CLOB file.
saveXslBufferToFile()	Saves the XSL buffer to file.
setHostname(String)	Sets database host name.
setInstancename(String)	Sets database instance name.
setPassword(String)	Sets user password.
setPort(String)	Sets database port number.
setResBuffer(String)	Sets new text in the result buffer.
setResCLOBFileName(String)	Sets Result CLOB file name.
setResCLOBTableName(String)	Sets Result CLOB table name.

Table 10–4 DBViewer Bean Methods (Cont.)

Method	Description
setResFileName(String)	Sets Result file name.
setResHtmlView(boolean)	Shows the result buffer as rendered HTML.
setResSourceEditView(boolean)	Shows the result buffer as XML source and enter edit mode.
setResSourceView(boolean)	Shows the result buffer as XML source.
setResTreeView(boolean)	Shows the result buffer as XML tree view.
setUsername(String)	Sets user name.
setXmlBuffer(String)	Sets new text in the XML buffer.
setXmlCLOBFileName(String)	Sets XML CLOB table name.
setXmlCLOBTableName(String)	Sets XML CLOB table name.
setXmlFileName(String)	Sets XML file name.
setXmlSourceEditView(boolean)	Shows the XML buffer as XML source and enter edit mode.
setXmlSourceView(boolean)	Shows the XML buffer as XML source.
setXmlTreeView(boolean)	Shows the XML buffer as tree.
setXslBuffer(String)	Sets new text in the XSL buffer.
setXslCLOBFileName(String)	Sets XSL CLOB file name.
setXslCLOBTableName(String)	Sets XSL CLOB table name.
setXslFileName(String)	Sets XSL file name.
setXslSourceEditView(boolean)	Shows the XSL buffer as XML source and enter edit mode.
setXslSourceView(boolean)	Shows the XSL buffer as XML source.
setXslTreeView(boolean)	Shows the XSL buffer as tree.
transformToDoc()	Transfroms the content of the XML buffer by applying the stylesheet from the XSL buffer.
transformToRes()	Applies the stylesheet transformation from the XSL buffer to the XML in the XML buffer and stores the result into the result buffer.
transformToString()	Transfroms the content of the XML buffer by applying the stylesheet from the XSL buffer.

Using DBAccess Bean

DBAccess bean maintains CLOB tables that can hold multiple XML and text documents. Each table is created using the following statement:

```
CREATE TABLE tablename FILENAME CHAR( 16) UNIQUE, FILEDATA CLOB) LOB(FILEDATA)
STORE AS (DISABLE STORAGE IN ROW)
```

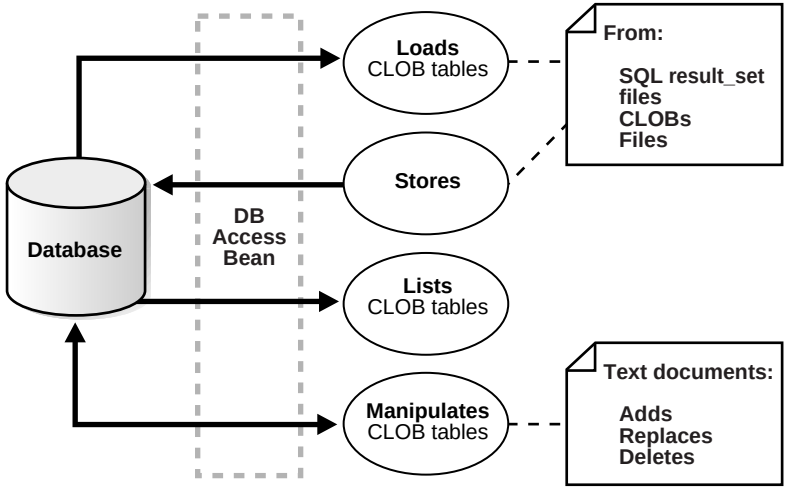
Each XML (or text) document is stored as a row in the table. The FILENAME field holds a unique string used as a key to retrieve, update, or delete the row. Document text is stored in the FILEDATA field. This is a CLOB object. CLOB tables are automatically maintained by the Transviewer bean. The CLOB tables maintained by DBAccess bean can be later used by the Transviewer bean. DBAccess bean does the following tasks:

- Creates and deletes CLOB tables
- Lists a CLOB table's contents
- Adds, replaces, or deletes text documents in the CLOB tables

DBAcess Bean Usage

Figure 10–11 illustrates the DBAccess bean usage. It shows how DBAccess bean maintains, and manipulates XML documents stored in CLOBs.

Figure 10–11 DBAccess Bean Usage Diagram



DBAccess Bean Methods

Table 10–5 lists the DBAccess bean methods.

Table 10–5 DBAccess Bean Methods

Method	Description
createXMLTable(Connection, String)	Creates XML table.
deleteXMLName(Connection, String, String)	Deletes text file from XML table.
dropXMLTable(Connection, String)	Deletes XML table.
getNameSize()	Returns the size of the field where the filename is kept.
getXMLData(Connection, String, String)	Retrieve text file from XML table.
getXMLNames(Connection, String)	Returns all file names in XML table.
getXMLTableNames(Connection, String)	Gets all XML tables with names starting with a given string.
insertXMLData(Connection, String, String, String)	Inserts text file as a row in XML table.
isXMLTable(Connection, String)	Checks if the table is XML table.
replaceXMLData(Connection, String, String, String)	Replaces text file as a row in XML table.
xmlTableExists(Connection, String)	Checks if XML table exists.

Using the XMLDiff Bean

The XML Diff Bean performs a tree comparison on two XML DOM trees. It displays the two XML trees and shows the differences between the XML trees. A node can be inserted, deleted, moved, or modified. Each of these operations is shown in a different color or style as in the following list:

- Red—Used to show a modified Node or Attribute
- Blue—Used to show a new Node or Attribute
- Black—Used to show a deleted Node or Attribute

Moves will be displayed visually as a delete or insert operation.

You can generate the differences between the two XML trees in the form of XSL code. The first XML file can be transformed into the second XML file by using the XSL code generated.

Note: Currently you cannot customize the GUI display.

XMLDiff Methods

The XMLDiff Bean has the methods described in this section.

boolean diff()

Finds the differences between the two XML files or the two XMLDocument objects.

void domBuilderError(DOMBuilderEvent p0)

Implements the DOMBuilderErrorListener interface called only by the DOM parser.

void domBuilderErrorCalled(DOMBuilderErrorEvent p0)

Implements the DOMBuilderErrorListener interface called only by the DOM parser when there is an error while parsing.

void domBuilderOver(DOMBuilderEvent p0)

Implements the DOMBuilderListener interface called only by a DOM parser thread when the parsing is done.

void domBuilderStarted(DOMBuilderEvent p0)

Implements the DOMBuilderListener interface called only by the DOM parser when the parsing begins.

boolean equals(Node node1, Node node2)

Performs the comparison of two nodes. It is called by the differ algorithm. You can overwrite this function for customized comparisons.

XMLDocument generateXSLDoc()

Generates an XSL stylesheet as an XMLDocument that initially represents the differences between the two XML document sets.

void generateXSLFile(java.lang.String filename)

Generates an XSL file of input filename that represents the differences between the two XML files which were initially set.

javax.swing.JTextPane getDiffPane1()

Gets the text panel as JTextPane object that visually shows the diffs in the first XML file.

javax.swing.JTextPane getDiffPane2()

Gets the text panel as a JTextPane object that visually shows the diffs in the second XML file or document.

XMLDocument getDocument1()

Gets the document root as an XMLDocument object of the first XML tree

XMLDocument getDocument2()

Gets the document root as an XMLDocument object of the second XML tree

void printDiffTree(int tree, BufferedWriter out)

Prints the diff tree that contains the node names and values that have been identified as diffs by the algorithm. This method is useful for debugging.

void setDocuments(XMLDocument doc1, XMLDocument doc2)

Sets the XML documents which need to be compared.

void setFiles(java.io.File file1, java.io.File file2)

Sets the XML files which need to be compared.

void setIndentIncr(int spaces)

Sets the indentation for the XSL generation. This should be called before the generateXSLFile() or generateXSLDoc() methods. The indentation will be applied to all attributes only. For indenting newly inserted nodes besides attributes see void setNewNodeIndentIncr(int spaces).

void setInput1(java.io.File file1)

Sets the first XML file that needs to be compared.

void setInput1(XMLDocument doc1)

Sets the first XML document that needs to be compared.

void setInput2(java.io.File file2)

Sets the second XML file that needs to be compared.

void setInput2(XMLDocument doc2)

Sets the second XML document that needs to be compared.

void setNewNodeIndentIncr(int spaces)

Sets the indentation for the XSL generation. This should be called before the generateXSLFile() or generateXSLDoc() methods. The indentation will be applied to all newly inserted nodes only (except attributes). For attributes indentation support see void setIndentIncr(int spaces).

void setNoMoves()

Assumes that there are no moves to be detected by the diff algorithm. This function should be called before the diff() function. Using this method should improve performance.

Running the Transviewer Bean Samples

The XDK for Java Transviewer bean sample/ directory contains sample Transviewer bean applications that illustrate how to use Oracle Transviewer beans. Oracle Transviewer beans toolset contains DOMBuilder, XMLSourceView,

XMLTreeView, XSLTransformer, XMLTransformPanel, DBViewer, DBAccess, and XMLDiff beans.

Table 10–6 lists the sample files in sample/.

Table 10–6 Transviewer Bean Sample Files

File Name	Description
booklist.xml	Sample XML file used by Example 1, 2, or 3.
doc.xml	Sample XML file used by Example 1, 2, or 3.
doc.html	Sample HTML file used by Examples 1, 2, or 3.
doc.xsl	Sample input XSL file used by Examples 1, 2, or 3. doc.xsl is used by XSLTransformer.
emptable.xsl	Sample input XSL file used by Examples 1, 2, or 3.
tohtml.xsl	Sample input XSL file used by Examples 1, 2, or 3. Transforms booklist.xml.
AsyncTransformSample.java See "Transviewer Bean Example 1: AsyncTransformSample.java".	Sample nonvisual application using XSLTransformer bean and DOMBuilder bean. It applies the XSLT stylesheet specified in doc.xsl on all *.xml files from the current directory. The results are in the files with extension.log.
ViewSample.java See "Transviewer Bean Example 2: ViewSample.java".	Sample visual application that uses XMLSourceView and XMLTreeView beans.It visualizes XML document files.

Table 10–6 Transviewer Bean Sample Files (Cont.)

File Name	Description
XMLTransformPanelSample.java See "Transviewer Bean Example 3: XMLTransformPanelSample.java".	A visual application that uses XMLTransformPanel bean. This bean uses all four beans from above. It applies XSL transformations on XML documents and shows the result Visualizes and allows editing of XML and XSL input files.
DBViewSample See: ■ "Transviewer Bean Example 4a: DBViewer Bean — DBViewClaims.java" ■ "Transviewer Bean Example 4b: DBViewer Bean — DBViewFrame.java" ■ "Transviewer Bean Example 4c: DBViewer Bean — DBViewSample.java"	A sample visual application that uses DBViewer bean to implement simple insurance claim handling application.
XMLDiffSample See: "XMLDiffSample.java" "XMLDiffFrame.java"	A sample visual application by which users can graphically compare any two XML files. The differences between the two files can be viewed as XSLT code. The first XML file can be transformed into the second XML file using the generated XSLT.

Installing the Transviewer Bean Samples

The Transviewer beans require as a minimum JDK 1.1.6, and can be used with any version of JDK 1.2.

1. Download and install the following components used by the Transviewer beans:
   - Oracle JDBC Driver for thin client (jar file classes111.zip)
   - Oracle XML SQL Utility (jar file oraclexmlsql.jar)

   After installing this components, include classes111.zip and oraclexmlsql.jar in your classpath.
2. The beans and the samples use swing 1.1. If you use jdk1.2, go to step 3. If you use jdk1.1, you will need to download Swing 1.1 from Sun. After downloading Swing, add swingall.jar to your CLASSPATH.
3. Change JDKPATH in `Makefile` to point to your JDK path. In addition, on Windows NT, change the file separator as stated in the `Makefile`. If you do not have an ORACLE_HOME set, then set it to the root directory of your XDK JavaBeans installation.

4. If you are not using the default database with a scott/tiger account, change USERID and PASSWORD in the Makefile to run Sample4
5. Run "**make**" to generate .class files.
6. Run the sample programs using commands:
   - **gmake** sample1
   - **gmake** sample2
   - **gmake** sample3
   - **gmake** sample4
   - **gmake** sample6
7. Visualize the results in .log files using the ViewSample.
8. Use the XSLT document from './tohtml.xsl' to transform the XML document from './booklist.xml'.

Use the sample files XMLDiffData1.txt and XMLDiffData2.txt to test the demo sample6 for the XMLDiff Bean. A few .xml files are provided as test cases. An XSL stylesheet 'doc.xsl' is used by XSLTransformer.

Note: sample1 runs the XMLTransViewer program so that you can import and export XML files from Oracle9*i*, keep your XSL transformation files in Oracle9*i*, and apply stylesheets to XML interactively.

Using Database Connectivity

To use the database connectivity feature in this program, you must know the following:

- Network name of the computer where Oracle9*i* or Oracle9*i* Application Server runs
- Port (usually 1521)
- Name of the oracle instance (usually orcl)

You also need an account with CREATE TABLE privilege.

You can try the default account scott with password tiger if it still enabled on your Oracle9i system.

Running Makefile

The following is the `makefile` script:

```
# Makefile for sample java files

.SUFFIXES : .java .class

CLASSES = ViewSample.class  AsyncTransformSample.class
XMLTransformPanelSample.class

# Change it to the appropriate separator based on the OS
PATHSEP= :

# Change this path to your JDK location. If you use JDK 1.1, you will need
# to download also Swing 1.1 and add swingall.jar to your classpath.
# You do not need to do this for JDK 1.2 since Swing is part of JDK 1.2
JDKPATH = /usr/local/packages/jdk1.2

# Make sure that the following product jar/zip files are in the classpath:
# - Oracle JDBC driver for thin client (file classes111.zip)
# - Oracle XML SQL Utility (file oraclexmlsql.jar)
# You can download this products from technet.us.oracle.com

#
CLASSPATH
:=$(CLASSPATH)$(PATHSEP)../lib/xmlparserv2.jar$(PATHSEP)../lib/xmlcomp.jar$(PATH
SEP)../lib/jdev-rt.zip$(PATHSEP).$(PATHSEP)
%.class: %.java
$(JDKPATH)/bin/javac -classpath "$(CLASSPATH)" $<

# make all class files
all: $(CLASSES)


sample1: XMLTransformPanelSample.class
$(JDKPATH)/bin/java -classpath "$(CLASSPATH)" XMLTransformPanelSample
sample2: ViewSample.class
$(JDKPATH)/bin/java -classpath "$(CLASSPATH)" ViewSample
sample3: AsyncTransformSample.class
$(JDKPATH)/bin/java -classpath "$(CLASSPATH)" AsyncTransformSample
```

Transviewer Bean Example 1: AsyncTransformSample.java

This example shows you how to use DOMBuilder and the XSLTransformer beans to asynchronously transform multiple XML files.

```
import java.net.URL;
import java.net.MalformedURLException;
import java.io.IOException;
import java.io.InputStream;
import java.io.ObjectInputStream;
import java.io.OutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.PrintWriter;
import java.util.Vector;

import org.w3c.dom.DocumentFragment;
import org.w3c.dom.DOMException;

import oracle.xml.async.DOMBuilder;
import oracle.xml.async.DOMBuilderEvent;
import oracle.xml.async.DOMBuilderListener;
import oracle.xml.async.DOMBuilderErrorEvent;
import oracle.xml.async.DOMBuilderErrorListener;
import oracle.xml.async.XSLTransformer;
import oracle.xml.async.XSLTransformerEvent;
import oracle.xml.async.XSLTransformerListener;
import oracle.xml.async.XSLTransformerErrorEvent;
import oracle.xml.async.XSLTransformerErrorListener;
import oracle.xml.async.ResourceManager;
import oracle.xml.parser.v2.DOMParser;
import oracle.xml.parser.v2.XMLDocument;
import oracle.xml.parser.v2.XSLStylesheet;
import oracle.xml.parser.v2.*;

public class AsyncTransformSample
{
  /**
   *  uses DOMBuilder bean
   */
   void runDOMBuilders ()
   {
      rm = new ResourceManager (numXMLDocs);

      for (int i = 0; i < numXMLDocs; i++)
```

```
    {
       rm.getResource();

       try
       {
          DOMBuilder builder = new DOMBuilder(i);

          URL  xmlURL = createURL(basedir + "/" +
                                  (String)xmlfiles.elementAt(i));
          if (xmlURL == null)
             exitWithError("File " + (String)xmlfiles.elementAt(i) +
                           " not found");

          builder.setPreserveWhitespace(true);
          builder.setBaseURL (createURL(basedir + "/"));
          builder.addDOMBuilderListener (new DOMBuilderListener() {
             public void domBuilderStarted(DOMBuilderEvent p0) {}
             public void domBuilderError(DOMBuilderEvent p0) {}
             public synchronized void domBuilderOver(DOMBuilderEvent p0)
             {
                DOMBuilder bld = (DOMBuilder)p0.getSource();
                runXSLTransformer (bld.getDocument(), bld.getId());
             }
          });
          builder.addDOMBuilderErrorListener (new DOMBuilderErrorListener() {
             public void domBuilderErrorCalled(DOMBuilderErrorEvent p0)
             {
                int id = ((DOMBuilder)p0.getSource()).getId();
                exitWithError("Error occurred while parsing " +
                   xmlfiles.elementAt(id) + ": " +
                   p0.getException().getMessage());
             }
          });
          builder.parse (xmlURL);

          System.err.println("Parsing file " + xmlfiles.elementAt(i));
       }
       catch (Exception e)
       {
          exitWithError("Error occurred while parsing " +
                        (String)xmlfiles.elementAt(i) + ": " +
                        e.getMessage());
       }
    }
 }
```

```
/**
 *  uses XSLTransformer bean
 */
 void runXSLTransformer (XMLDocument xml, int id)
 {
    try
    {
       XSLTransformer processor = new XSLTransformer (id);
       XSLStylesheet  xsl       = new XSLStylesheet (xsldoc, xslURL);

       processor.showWarnings (true);
       processor.setErrorStream (errors);
       processor.addXSLTransformerListener (new XSLTransformerListener() {
          public void xslTransformerStarted (XSLTransformerEvent p0) {}
          public void xslTransformerError(XSLTransformerEvent p0) {}
          public void xslTransformerOver (XSLTransformerEvent p0)
          {
             XSLTransformer trans = (XSLTransformer)p0.getSource();
             saveResult (trans.getResult(),  trans.getId());
          }
       });
 processor.addXSLTransformerErrorListener (new XSLTransformerErrorListener() {
 public void xslTransformerErrorCalled(XSLTransformerErrorEvent p0)
          {
             int i = ((XSLTransformer)p0.getSource()).getId();
           exitWithError("Error occurred while processing " +
                   xmlfiles.elementAt(i) + ": " +
                   p0.getException().getMessage());
            }
       });
       processor.processXSL (xsl, xml);
       // transform xml document
    }
    catch (Exception e)
    {
       exitWithError("Error occurred while processing " + xslFile + ": " +
                     e.getMessage());
    }
 }

 void saveResult (DocumentFragment result, int id)
 {
   System.err.println("Transforming '" + xmlfiles.elementAt(id) +
            "' to '" + xmlfiles.elementAt(id) + ".log'" +
```

```
            " applying '" + xslFile);

   try
   {
       File resultFile = new File((String)xmlfiles.elementAt(id) + ".log");

       ((XMLNode)result).print(new FileOutputStream(resultFile));
   }
   catch (Exception e)
   {
      exitWithError("Error occurred while generating output : " +
                    e.getMessage());
   }

   rm.releaseResource();
}

void makeXSLDocument ()
{
   System.err.println ("Parsing file " + xslFile);
   try
   {
      DOMParser parser = new DOMParser();
      parser.setPreserveWhitespace (true);
      xslURL = createURL (xslFile);
      parser.parse (xslURL);
      xsldoc = parser.getDocument();
   }
   catch (Exception e)
   {
      exitWithError("Error occurred while parsing " + xslFile + ": " +
                    e.getMessage());
   }
}

private URL createURL(String fileName) throws Exception
{
   URL url = null;

   try
   {
      url = new URL(fileName);
   }
   catch (MalformedURLException ex)
   {
```

```
      File f = new File(fileName);

      try
      {
         String path = f.getAbsolutePath();
         // This is a bunch of weird code that is required to
         // make a valid URL on the Windows platform, due
         // to inconsistencies in what getAbsolutePath returns.
         String fs = System.getProperty("file.separator");
         if (fs.length() == 1)
         {
            char sep = fs.charAt(0);
            if (sep != '/')
               path = path.replace(sep, '/');
            if (path.charAt(0) != '/')
               path = '/' + path;
         }
         path = "file://" + path;
         url = new URL(path);
      }
      catch (MalformedURLException e)
      {
         exitWithError("Cannot create url for: " + fileName);
      }
   }

   return url;
}

boolean init () throws Exception
{
   File     directory = new File (basedir);
   String[] dirfiles = directory.list();
   for (int j = 0; j < dirfiles.length; j++)
   {
      String dirfile = dirfiles[j];

      if (!dirfile.endsWith(".xml"))
          continue;

       xmlfiles.addElement(dirfile);
   }

   if (xmlfiles.isEmpty()) {
   System.out.println("No files in directory were selected for processing");
```

```
      return false;
   }
   numXMLDocs = xmlfiles.size();

   return true;
}

private void exitWithError(String msg)
{
   PrintWriter errs = new PrintWriter(errors);
   errs.println(msg);
   errs.flush();
   System.exit(1);
}

void asyncTransform () throws Exception
{
   System.err.println (numXMLDocs +
            " XML documents will be transformed" +
            " using XSLT stylesheet specified in " + xslFile +
            " with " +  numXMLDocs + " threads");

   makeXSLDocument ();
   runDOMBuilders ();

   // wait for the last request to complete
   while (rm.activeFound())
      Thread.sleep(100);

}
String       basedir = new String (".");
OutputStream errors = System.err;

Vector xmlfiles = new Vector();
int    numXMLDocs = 1;

String      xslFile = new String ("doc.xsl");
URL         xslURL;
XMLDocument xsldoc;

private ResourceManager rm;

/**
 *   main
 */
```

```
   public static void main (String args[])
   {
      AsyncTransformSample inst = new AsyncTransformSample();

      try
      {
         if (!inst.init())
            System.exit(0);

         inst.asyncTransform ();
      }
      catch (Exception e)
      {
         e.printStackTrace();
      }

      System.exit(0);
   }
}
```

Transviewer Bean Example 2: ViewSample.java

This example shows you how to use XMLSourceView and XMLTreeView beans to visually represent XML files.

```
import java.awt.*;
import oracle.xml.srcviewer.*;
import oracle.xml.treeviewer.*;
import oracle.xml.parser.v2.XMLDocument;
import oracle.xml.parser.v2.*;
import org.w3c.dom.*;
import java.net.*;
import java.io.*;
import java.util.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

public class ViewSample
{

  public static void main(String[] args)
  {
     String fileName = new String ("booklist.xml");
     if (args.length > 0) {
```

```
      fileName = args[0];
    }

    JFrame        frame         = setFrame ("XMLViewer");
    XMLDocument   xmlDocument   = getXMLDocumentFromFile (fileName);
    XMLSourceView xmlSourceView = setXMLSourceView (xmlDocument);
    XMLTreeView   xmlTreeView   = setXMLTreeView (xmlDocument);
    JTabbedPane   jtbPane       = new JTabbedPane ();

    jtbPane.addTab ("Source", null, xmlSourceView, "XML document sorce view");
    jtbPane.addTab ("Tree", null, xmlTreeView, "XML document tree view");
    jtbPane.setPreferredSize (new Dimension(400,300));
    frame.getContentPane().add (jtbPane);

    frame.setTitle    (fileName);
    frame.setJMenuBar (setMenuBar());
    frame.setVisible  (true);
}

static JFrame setFrame (String title)
{
  JFrame frame = new JFrame (title);
  //Center the window
  Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
  Dimension frameSize  = frame.getSize();
  if (frameSize.height > screenSize.height) {
     frameSize.height = screenSize.height;
  }
  if (frameSize.width > screenSize.width) {
    frameSize.width = screenSize.width;
  }
  frame.setLocation ((screenSize.width - frameSize.width)/2,
                     (screenSize.height - frameSize.height)/2);
  frame.addWindowListener(new WindowAdapter() {
     public void windowClosing(WindowEvent e) {
        System.exit(0);
     }
  });
  frame.getContentPane().setLayout (new BorderLayout());
  frame.setSize(new Dimension(400, 300));
  frame.setVisible (false);
  frame.setTitle (title);

  return frame;
}
```

```
static JMenuBar setMenuBar ()
{
   JMenuBar menuBar = new JMenuBar();
   JMenu    menu    = new JMenu ("Exit");
   menu.addMenuListener ( new MenuListener () {
     public void menuSelected (MenuEvent ev) { System.exit(0); }
     public void menuDeselected (MenuEvent ev) {}
     public void menuCanceled (MenuEvent ev) {}
   });
   menuBar.add (menu);
   return menuBar;
}

/**
 * creates  XMLSourceView object
 */
static XMLSourceView setXMLSourceView(XMLDocument xmlDocument)
{
  XMLSourceView xmlView = new XMLSourceView();

  xmlView.setXMLDocument(xmlDocument);
  xmlView.setBackground(Color.yellow);
  xmlView.setEditable(true);
  return xmlView;
}
/**
 * creates  XMLTreeView object
 */
static XMLTreeView setXMLTreeView(XMLDocument xmlDocument)
{
  XMLTreeView xmlView = new XMLTreeView();

  xmlView.setXMLDocument(xmlDocument);
  xmlView.setBackground(Color.yellow);
  return xmlView;
}

static XMLDocument getXMLDocumentFromFile (String fileName)
{
  XMLDocument doc = null;

  try  {
    DOMParser parser = new DOMParser();
    try {
```

```
            String dir= "" ;
            FileInputStream in = new FileInputStream(fileName);
            parser.setPreserveWhitespace(false);
            parser.setBaseURL(createURL(dir));
            parser.parse(in);
            in.close();
         } catch (Exception ex) {
            ex.printStackTrace();
            System.exit(0);
         }

         doc = (XMLDocument)parser.getDocument();

         try {
            doc.print(System.out);
         } catch (Exception ie) {
            ie.printStackTrace();
            System.exit(0);
         }

       }
       catch (Exception e) {
         e.printStackTrace();
       }
       return doc;
     }

     static URL createURL(String fileName)
     {
         URL url = null;
         try
         {
            url = new URL(fileName);
         }
         catch (MalformedURLException ex)
         {
            File f = new File(fileName);
            try
            {
               String path = f.getAbsolutePath();
               String fs = System.getProperty("file.separator");
               if (fs.length() == 1)
               {
                  char sep = fs.charAt(0);
                  if (sep != '/')
```

```
                path = path.replace(sep, '/');
             if (path.charAt(0) != '/')
                path = '/' + path;
          }
          path = "file://" + path;
          url = new URL(path);
       }
       catch (MalformedURLException e)
       {
          System.out.println("Cannot create url for: " + fileName);
          System.exit(0);
       }
    }
    return url;
  }
}
```

Transviewer Bean Example 3: XMLTransformPanelSample.java

This example is an interactive application that uses XMLTransformPanel bean to do the following:

- Generate XML from database queries
- Transform the XML using XSL stylesheets
- View the results
- Store the results in CLOB tables in the database

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import oracle.xml.transviewer.XMLTransformPanel;


public class XMLTransformPanelSample
{
  XMLTransformPanel transformPanel = new XMLTransformPanel();

  /**
   * Adjust frame size and add transformPanel to it.
   */
  public XMLTransformPanelSample ()
  {
    Frame     frame      = new JFrame();
```

```
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    frame.setSize(510,550);
    transformPanel.setPreferredSize(new Dimension(510,550));
    Dimension frameSize = frame.getSize();

    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
    }
    frame.setLocation ((screenSize.width - frameSize.width)/2,
                       (screenSize.height - frameSize.height)/2);
    frame.addWindowListener(new WindowAdapter() {
       public void windowClosing(WindowEvent e) { System.exit(0); }
    });
    frame.setVisible(true);

    ((JFrame)frame).getContentPane().add (transformPanel);
    frame.pack();
  }

  /**
   *  main(). Only creates XMLTransformPanelSample object.
   */
  public static void main (String[] args)
  {
    new XMLTransformPanelSample ();
  }
}
```

Transviewer Bean Example 4a: DBViewer Bean — DBViewClaims.java

This is an interactive example which lets you input the name or policy of an insurance claim. The appropriate claim is loaded as an XML buffer from the result set of an XML query. An XSL stylesheet is loaded from the file system. The DBViewer bean transforms the XML buffer using the XSL stylesheet to HTML. This HTML output can then be viewed.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import oracle.jdeveloper.layout.*;
import oracle.xml.dbviewer.*;
```

```
public class DBViewClaims extends JPanel {
  DBViewer dbPanel= new DBViewer();
  JButton searchButton = new JButton();
  XYLayout xYLayout1 = new XYLayout();
  JLabel titleLabel = new JLabel();
  JLabel nameLabel = new JLabel();
  JLabel policyLabel = new JLabel();
  JTextField nameTF = new JTextField();
  JTextField policyTF = new JTextField();
  JButton viewXMLButton = new JButton();
  JButton viewXSLButton = new JButton();
  JButton viewHTMLButton = new JButton();
  public DBViewClaims() {
    super();
    try  {
      jbInit();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
  private void jbInit() throws Exception {
    setBackground(SystemColor.controlLtHighlight);
    this.setLayout(xYLayout1);
    searchButton.setText("searchButton");
    searchButton.setLabel("Search");
    xYLayout1.setHeight(464);
    xYLayout1.setWidth(586);
    titleLabel.setText("List of Claims");
    titleLabel.setHorizontalAlignment(SwingConstants.CENTER);
    titleLabel.setBackground(new Color(192, 192, 255));
    titleLabel.setFont(new Font("Dialog", 1, 16));
    nameLabel.setText("Last Name");
    policyLabel.setText("Policy:");
    viewXMLButton.setText("viewXMLButton");
    viewXMLButton.setLabel("view XML");
    viewXMLButton.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        viewXMLButton_actionPerformed(e);
      }
    });
    viewXSLButton.setText("viewXSLButton");
    viewXSLButton.setLabel("view XSL");
    viewXSLButton.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
```

```
      viewXSLButton_actionPerformed(e);
    }
  });
  viewHTMLButton.setText("viewHTMLButton");
  viewHTMLButton.setLabel("view HTML");
  viewHTMLButton.addActionListener(new java.awt.event.ActionListener() {

    public void actionPerformed(ActionEvent e) {
      viewHTMLButton_actionPerformed(e);
    }
  });

   searchButton.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        searchButton_actionPerformed(e);
      }
   });

  this.add(dbPanel, new XYConstraints(16, 55, 552, 302));
  this.add(searchButton, new XYConstraints(413, 415, 154, 29));
  this.add(titleLabel, new XYConstraints(79, 10, 413, 31));
  this.add(nameLabel, new XYConstraints(333, 373, 72, -1));
  this.add(policyLabel, new XYConstraints(334, 395, 59, -1));
  this.add(nameTF, new XYConstraints(413, 368, 155, -1));
  this.add(policyTF, new XYConstraints(413, 391, 156, -1));
  this.add(viewXMLButton, new XYConstraints(19, 359, 94, 29));
  this.add(viewXSLButton, new XYConstraints(19, 390, 94, 29));
  this.add(viewHTMLButton, new XYConstraints(19, 421, 94, 29));
  updateUI();
}
void searchButton_actionPerformed(ActionEvent e) {
  String sqlText="select * from s_claim c ";
  try {
    if (!nameTF.getText().equals("")) {
       sqlText=sqlText+" where c.claimpolicy.primaryinsured.lastname="+
               "'"+nameTF.getText()+"'";
    } else if (!policyTF.getText().equals("")) {
       sqlText=sqlText+" where c.claimpolicy.policyid="+
               policyTF.getText();
    }
    dbPanel.setUsername("scott");
    dbPanel.setPassword("tiger");
    dbPanel.setInstancename("orcl");
    dbPanel.setHostname("localhost");
    dbPanel.setPort("1521");
```

```
      dbPanel.loadXMLBufferFromSQL(sqlText);
      dbPanel.loadXslBuffer("xslfiles","CLAIM.XSL");
      dbPanel.transformToRes();
      dbPanel.setResHtmlView(true);
    } catch (Exception e1) {
      System.out.println(e1);
    }
  }
  void viewXMLButton_actionPerformed(ActionEvent e) {
    dbPanel.setXmlSourceEditView(true);
  }
  void viewXSLButton_actionPerformed(ActionEvent e) {
    dbPanel.setXslSourceEditView(true);
  }
  void viewHTMLButton_actionPerformed(ActionEvent e) {
    dbPanel.setResHtmlView(true);
  }
}
```

Transviewer Bean Example 4b: DBViewer Bean — DBViewFrame.java

This example provides a frame with a menu bar to access the DBView Claims functionality. Claims can then be loaded and displayed in HTML.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import oracle.jdeveloper.layout.*;

public class DBViewFrame extends JFrame {
  JMenuBar menuBar1 = new JMenuBar();
  JMenu menuFile = new JMenu();
  JMenuItem menuFileExit = new JMenuItem();
  JMenuItem menuListCustomerClaims = new JMenuItem();

  public DBViewFrame() {
    super();
    try  {
      jbInit();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
  private void jbInit() throws Exception {
```

```
    this.getContentPane().setLayout(new GridLayout(1,1));
    this.setSize(new Dimension(600, 550));
    menuFile.setText("File");
    menuFileExit.setText("Exit");
    menuListCustomerClaims.setText("List Claims");
    menuFileExit.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        fileExit_ActionPerformed(e);
      }
    });
    menuListCustomerClaims.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
         ListCustomerClaims_ActionPerformed(e);

      }
    });
    menuFile.add(menuFileExit);
    menuFile.add(menuListCustomerClaims);
    menuBar1.add(menuFile);
    this.setJMenuBar(menuBar1);
    this.setBackground(SystemColor.controlLtHighlight);
  }
  void fileExit_ActionPerformed(ActionEvent e) {
    System.exit(0);
  }
  void ListCustomerClaims_ActionPerformed(ActionEvent e) {
    this.getContentPane().removeAll();
    this.getContentPane().add(new DBViewClaims());
    this.getContentPane().paintAll(this.getGraphics());
  }
}
```

Transviewer Bean Example 4c: DBViewer Bean — DBViewSample.java

This example simply provides a main function which instantiates DBViewFrame, giving it a specific look and feel.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class DBViewSample {
  public DBViewSample() {
    DBViewFrame frame = new DBViewFrame();
    frame.setVisible(true);
  }
```

```
  public static void main(String[] args) {
    try  {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    }
    catch (Exception e) {
      e.printStackTrace();
    }
    new DBViewSample();
  }
}
```

XMLDiffSample.java

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

import java.io.*;
import java.awt.*;
import javax.swing.*;
import javax.swing.tree.*;
import java.net.URL;
import java.net.MalformedURLException;

public class XMLDiffSample
{
  /**
   * Constructor
   */
  public XMLDiffSample() {
  }

  /**
   * main
   * @param args
   */
  public static void main(String[] args)
  {

    dfxApp = new XMLDiffSample();
    diffFrame = new XMLDiffFrame(dfxApp);
    diffFrame.addTransformMenu();
    xmlDiff = new XMLDiff();

    if (args.length == 3)
```

```
    outFile = args[2];
  /* Use the default outFile name = XMLDiffSample.xsl */
  if(args.length >= 2)
    dfxApp.showDiffs(new File(args[0]), new File(args[1]));

  diffFrame.setVisible(true);
}


 public void showDiffs(File file1, File file2)
 {
  try
  {
     xmlDiff.setFiles(file1, file2);

     /* Check if files are equal */
     if(!xmlDiff.diff())
     {
       JOptionPane.showMessageDialog(diffFrame,
         "Files are equivalent in XML representation",
         "XMLDiffSample Message",
         JOptionPane.PLAIN_MESSAGE);
     }

    /* generate xsl file */
   xmlDiff.generateXSLFile(outFile);
   /* parse the xsl file created, alternately you can use
      generateXSLDoc to get the xsl as a document tree instead of a file */
   parseXSL();
   /* Display the document trees created by the xmlDiff object */
   diffFrame.makeSrcPane(xmlDiff.getDocument1(), xmlDiff.getDocument2());
   diffFrame.makeDiffSrcPane(new XMLDiffSrcView(xmlDiff.getDiffPane1()),
                             new XMLDiffSrcView(xmlDiff.getDiffPane2()));
   diffFrame.makeXslPane(xslDoc, "Diff XSL Script");
   diffFrame.makeXslTabbedPane();
   }catch (FileNotFoundException e)
   {
     JOptionPane.showMessageDialog(diffFrame,
         "File Not Found: "+e.getMessage(),
         "XMLDiffSample Error Message",
         JOptionPane.ERROR_MESSAGE);
   }
   catch (Exception e)
   {
     e.printStackTrace();
```

```
        JOptionPane.showMessageDialog(diffFrame,
            "Error: "+e.getMessage(),
            "XMLDiffSample Error Message",
            JOptionPane.ERROR_MESSAGE);
      }
    }

    public void doXSLTransform()
    {
      try
      {
        doc1 = xmlDiff.getDocument1();
        doc2 = xmlDiff.getDocument2();

        XSLProcessor xslProc = new XSLProcessor();

        /* Using the xsl stylesheet generated (xslDoc), transform the first file
          (doc1) into the second file (resultDocFrag) */
         XMLDocumentFragment resultDocFrag = xslProc.processXSL(new XSLStylesheet
                                        (xslDoc, createURL(outFile)), doc1);
        XMLDocument resultDoc = new XMLDocument();
        /* The XML declaration has to be copied over to the transformed XML doc,
           the xsl will not generate it automatically */
        if (doc1.getFirstChild() instanceof XMLDeclPI)
        if (doc1.getFirstChild() instanceof XMLDeclPI)
        {
           XMLNode xmldecl = (XMLNode) resultDoc.importNode(doc1.getFirstChild(),
                                                            false);
           resultDoc.appendChild(xmldecl);
        }
       /* Create the DTD node in the transformed XML document  */
        if(doc1.getDoctype() != null)
        {
          DTD dtd = (DTD)doc1.getDoctype();
          resultDoc.setDoctype(dtd.getName(), dtd.getSystemId(),
dtd.getPublicId());
        }
        /* Create the result document tree from the document fragment */
        resultDoc.appendChild(resultDocFrag);
        diffFrame.makeResultFilePane(resultDoc);
      } catch (XSLException e)
      {
        e.printStackTrace();
        JOptionPane.showMessageDialog(diffFrame,
            "Error: "+e.getMessage(),
```

```
        "XMLDiffSample Error Message",
        JOptionPane.ERROR_MESSAGE);
    }
    catch (Exception e)
    {
      e.printStackTrace();
      JOptionPane.showMessageDialog(diffFrame,
       "Error:"+e.getMessage(),
       "XMLDiffSample Error Message",
       JOptionPane.ERROR_MESSAGE);
    }
  }

  /* Parse the XSL file generated into a DOM tree */
  protected void parseXSL()
  {
    try
    {
      BufferedReader xslFile = new BufferedReader(new FileReader(outFile));
      DOMParser domParser = new DOMParser();
      domParser.parse(xslFile);
      xslDoc = domParser.getDocument();

    }catch (FileNotFoundException e)
    {
      JOptionPane.showMessageDialog(diffFrame,
          "File Not Found: "+e.getMessage(),
          "XMLDiffSample Message",
          JOptionPane.PLAIN_MESSAGE);
    }
    catch (Exception e)
    {
      JOptionPane.showMessageDialog(diffFrame,
       "Error:"+e.getMessage(),
       "XMLDiffSample Error Message",
       JOptionPane.ERROR_MESSAGE);
    }
  }

  // create a URL from a file name
  protected URL createURL(String fileName)
  {
      URL url = null;
      try
      {
```

```
            url = new URL(fileName);
         }
         catch (MalformedURLException ex)
         {
            File f = new File(fileName);
            try
            {
               String path = f.getAbsolutePath();
               // to handle Windows platform
               String fs = System.getProperty("file.separator");
               if (fs.length() == 1)
               {
                  char sep = fs.charAt(0);
                  if (sep != '/')
                     path = path.replace(sep, '/');
                  if (path.charAt(0) != '/')
                     path = '/' + path;
               }
               path = "file://" + path;
               url = new URL(path);
            }
            catch (MalformedURLException e)
            {
             JOptionPane.showMessageDialog(diffFrame,
             "Cannot create url for: " + fileName,
             "XMLDiffSample Error Message",
             JOptionPane.ERROR_MESSAGE);

            }
         }
         return url;
      }

      protected XMLDocument doc1;   /* DOM tree for first file */
      protected XMLDocument doc2;   /* DOME 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 */
   }
```

XMLDiffFrame.java

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;

import oracle.xml.parser.v2.*;
import oracle.xml.srcviewer.*;
import oracle.xml.differ.*;
import org.w3c.dom.*;

public class XMLDiffFrame extends JFrame implements ActionListener {

  public XMLDiffFrame(XMLDiffSample dfApp)
  {
    super();
    mydfApp = dfApp;
    init();
  }


  public void makeSrcPane(XMLDocument doc1, XMLDocument doc2)
  {
      //undo srcviewer highlighting here
      XMLSourceView XmlSrcView1 = new XMLSourceView();
      XmlSrcView1.setXMLDocument(doc1);
      XmlSrcView1.setTagForeground(Color.black);
      XmlSrcView1.setAttributeValueForeground(Color.black);
      XmlSrcView1.setPIDataForeground(Color.black);
      XmlSrcView1.setCommentDataForeground(Color.black);
      XmlSrcView1.setCDATAForeground(Color.black);

      XmlSrcView1.setBackground(Color.lightGray);
      XmlSrcView1.getJTextPane().setBackground(Color.white);
      XmlSrcView1.add(new JLabel(filename1,SwingConstants.CENTER),
                      BorderLayout.NORTH);

      XMLSourceView XmlSrcView2 = new XMLSourceView();
      XmlSrcView2.setXMLDocument(doc2);
      XmlSrcView2.setTagForeground(Color.black);
      XmlSrcView2.setAttributeValueForeground(Color.black);
      XmlSrcView2.setPIDataForeground(Color.black);
      XmlSrcView2.setCommentDataForeground(Color.black);
      XmlSrcView2.setCDATAForeground(Color.black);
```

```
      XmlSrcView2.setBackground(Color.lightGray);
      XmlSrcView2.getJTextPane().setBackground(Color.white);
      XmlSrcView2.add(new JLabel(filename2,SwingConstants.CENTER),
                      BorderLayout.NORTH);

      XmlSrcView2.updateUI();
      XmlSrcView1.updateUI();

      srcPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
                                          XmlSrcView1, XmlSrcView2);
      srcPane.setSize(FRAMEWIDTH,FRAMEHEIGHT);
      srcPane.setDividerLocation(0.5);
      srcPane.validate();


  }

  public void makeDiffSrcPane(XMLDiffSrcView srcView1, XMLDiffSrcView srcView2)
  {
      srcView1.setBackground(Color.lightGray);
      srcView2.setBackground(Color.lightGray);

      srcView1.add(new
JLabel(filename1,SwingConstants.CENTER),BorderLayout.NORTH);
      srcView2.add(new
JLabel(filename2,SwingConstants.CENTER),BorderLayout.NORTH);

      JScrollBar vscrollBar = srcView2.getScrollPane().getVerticalScrollBar();

      // make the diffSrcView divider fixed.
      srcView1.getScrollPane().setVerticalScrollBar(vscrollBar);
      srcView1.getScrollPane().setMinimumSize(
        new
Dimension(FRAMEWIDTH/2,srcView1.getScrollPane().getPreferredSize().height));
      srcView2.getScrollPane().setMinimumSize(
        new
Dimension(FRAMEWIDTH/2,srcView2.getScrollPane().getPreferredSize().height));

      srcView2.getScrollPane().updateUI();
      srcView1.getScrollPane().updateUI();

      srcView2.getTextPane().updateUI();
      srcView1.getTextPane().updateUI();
```

```
      srcView2.updateUI();
      srcView1.updateUI();

      diffSrcPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
                                           srcView1, srcView2);
      diffSrcPane.setSize(FRAMEWIDTH,FRAMEHEIGHT);
      diffSrcPane.setDividerLocation(0.5);
      diffSrcPane.validate();

  }
  public void makeTabbedPane()
  {
    tabbedPane = new JTabbedPane();

    tabbedPane.addTab("SourceView", null , srcPane, "Source View of Files being
Diffed");
    tabbedPane.addTab("SourceDiffView", null , diffSrcPane, "Source View of File
Diffs");
    tabbedPane.addTab("TreeDiffView", null , diffTreePane, "DOM Tree View of
File Diffs");
    tabbedPane.setSelectedIndex(1);
    tabbedPane.setSize(FRAMEWIDTH,FRAMEHEIGHT);

    this.getContentPane().add(tabbedPane);
    this.setVisible(true);


  }

  public void makeXslPane(XMLDocument doc, String title)
  {
     xslSrcView = new XMLSourceView();
     xslSrcView.setXMLDocument(doc);
     xslSrcView.setTagForeground(Color.black);
     xslSrcView.setAttributeValueForeground(Color.black);
     xslSrcView.setPIDataForeground(Color.black);
     xslSrcView.setCommentDataForeground(Color.black);
     xslSrcView.setCDATAForeground(Color.black);

     xslSrcView.setBackground(Color.lightGray);
     xslSrcView.getJTextPane().setBackground(Color.white);
     xslSrcView.add(new JLabel(title,SwingConstants.CENTER),
                     BorderLayout.NORTH);
     this.enableTransformItem(true);
  }
```

```
  public void makeResultFilePane(XMLDocument doc)
  {
    resultDoc = doc;
    XMLSourceView resultSrcView = new XMLSourceView();
    resultSrcView.setXMLDocument(doc);
    resultSrcView.setTagForeground(Color.black);
    resultSrcView.setAttributeValueForeground(Color.black);
    resultSrcView.setPIDataForeground(Color.black);
    resultSrcView.setCommentDataForeground(Color.black);
    resultSrcView.setCDATAForeground(Color.black);

    resultSrcView.setBackground(Color.lightGray);
    resultSrcView.getJTextPane().setBackground(Color.white);
    resultSrcView.add(new JLabel("XSLT Result File",SwingConstants.CENTER),
                      BorderLayout.NORTH);

    tabbedPane.addTab("ResultSourceView", null , resultSrcView,
                         "Source View of XSLT on File1");
    tabbedPane.setSelectedIndex(3);
    this.enableSaveAsItem(true);
  }

  public void makeXslTabbedPane()
  {
    tabbedPane = new JTabbedPane();

    tabbedPane.addTab("SourceView", null , srcPane, "Source View of XML Files
being Diffed");
    tabbedPane.addTab("SourceDiffView", null , diffSrcPane, "Source View of File
Diffs");
    tabbedPane.addTab("XSL Script",null,xslSrcView, "Source View of Diff XSL
script");
    tabbedPane.setSelectedIndex(2);
    tabbedPane.setSize(FRAMEWIDTH,FRAMEHEIGHT);

    this.getContentPane().add(tabbedPane);
    this.setVisible(true);


  }

  public void actionPerformed(ActionEvent evt)
  {
```

```
File selectedFile1, selectedFile2;
BufferedReader file1, file2;
String arg, temp;

if(evt.getSource() instanceof JMenuItem)
{

  arg = evt.getActionCommand();

  if(arg.equals("Compare XML Files"))
  {
    JFileChooser jFC = new JFileChooser();
  jFC.setCurrentDirectory(new File("."));
  int retval = jFC.showOpenDialog(this);

  switch (retval)
  {

    case JFileChooser.APPROVE_OPTION:
        selectedFile1 = jFC.getSelectedFile();
      temp = selectedFile1.getName();
        jFC.cancelSelection();
        jFC.updateUI();
      switch(jFC.showOpenDialog(this))
      {
        case JFileChooser.APPROVE_OPTION:
          selectedFile2 = jFC.getSelectedFile();
         filename2 = selectedFile2.getName();
          filename1 = temp;

          this.getContentPane().removeAll();
          this.enableSaveAsItem(false);

         mydfApp.showDiffs(selectedFile1, selectedFile2);
          break;

          case JFileChooser.CANCEL_OPTION:
         break; //filename1 = null; // filename1 also null
        }// switch (jFC.showOpenDialog(this))
      break;

      case JFileChooser.CANCEL_OPTION:
      break;
    }
  }// if(arg.equals("Compare XML Files"))
```

```
    else if(arg.equals("Apply XSL to 1st Input File"))
    {
      mydfApp.doXSLTransform();

    }
    else if(arg.equals("Save As"))
    {
      JFileChooser jFC = new JFileChooser();
      jFC.setCurrentDirectory(new File("."));
      int retval = jFC.showOpenDialog(this);

    if (retval == JFileChooser.APPROVE_OPTION)
      {
        File file = jFC.getSelectedFile();
        try
        {
          resultDoc.print(new FileOutputStream(file));
        }catch (IOException e)
        {
           JOptionPane.showMessageDialog(this,
           "Error:"+e.getMessage(),
           "XMLDiffer Message",
           JOptionPane.PLAIN_MESSAGE);
        }


      }
    }
    else if(arg.equals("Exit"))
    {
      System.exit(0);
    }

  }
}


public void addTransformMenu()
{
  JMenuItem  item;

  JMenu jmenu = new JMenu("Transform");

  item = new JMenuItem("Apply XSL to 1st Input File");
  item.addActionListener(this);
```

```
    item.setEnabled(false);
    jmenu.add(item);

    this.getJMenuBar().add(jmenu);

  }

  protected void enableTransformItem(boolean flag)
  {
    this.getJMenuBar().getMenu(1).getItem(0).setEnabled(flag);
  }

  protected void enableSaveAsItem(boolean flag)
  {
    this.getJMenuBar().getMenu(0).getItem(1).setEnabled(flag);
  }

  private void init()
  {
    try
    {
      this.setTitle("XMLDiffer");
      this.getContentPane().setLayout( new
BoxLayout(this.getContentPane(),BoxLayout.Y_AXIS));
      // make the Differ window non-resizable
      this.setResizable(false);
      this.getContentPane().setBackground(SystemColor.control);
      addMenu();

      Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
      Dimension frameSize = this.getSize();

      // set Frame size based on screen size such that there is room around it
      FRAMEWIDTH = screenSize.width - 100;
      FRAMEHEIGHT = screenSize.height - 200;
      this.setSize(new Dimension(FRAMEWIDTH, FRAMEHEIGHT));

      // put Differ window in the center of the screen
      this.setLocation((screenSize.width - FRAMEWIDTH)/2, (screenSize.height -
FRAMEHEIGHT)/2);
      this.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) { System.exit(0); }});

    }
    catch (Exception e)
```

```
  {
    e.printStackTrace();
  }
}

private void addMenu()
{
  JMenuItem  item;

  JMenuBar jmenubar = new JMenuBar();
  JMenu jmenu = new JMenu("File");

  item = new JMenuItem("Compare XML Files");
  item.addActionListener(this);
  jmenu.add(item);

  item = new JMenuItem("Save As");
  item.addActionListener(this);
  item.setEnabled(false);
  jmenu.add(item);

  jmenu.addSeparator();

  item = new JMenuItem("Exit");
  item.addActionListener(this);
  jmenu.add(item);

  jmenubar.add(jmenu);
  this.setJMenuBar(jmenubar);

}


protected static int LEFT_TOP = 0;
protected static int RIGHT_TOP = 1;
protected static int CENTER = 2;

private int FRAMEWIDTH =0;
private int FRAMEHEIGHT =0;

private XMLDocument resultDoc;
private XMLSourceView xslSrcView;
private XMLDiffSample mydfApp;
private String filename1, filename2;
private JTabbedPane tabbedPane;
```

```
  private JSplitPane diffTreePane, srcPane,diffSrcPane;
}
```

11

Using XDK and SOAP

This chapter contains the following sections:

What Is SOAP?

The term *Web services* is used for the functionality made available by an entity over the Web. It is an application that uses XML standards and is published, located and executed through the Web.

The Simple Object Access Protocol (SOAP) is a protocol for sending and receiving requests and responses across the Internet. Because it is based on XML and simple transport protocols such as HTTP, it is not blocked by firewalls and is very easy to use. SOAP is independent of operating system, independent of implementation language, and independent of any single object model.

SOAP supports remote procedure calls. Its messages are only of the three types:

- A request for a service, including input parameters
- A response to the requested service, including return value and output parameters
- A fault containing error codes and information

SOAP messages consist of:

- an *envelope* that contains the message, defines how to process the message, who should process the message, and whether processing is optional or mandatory.
- *encoding rules* that describe the data types for the application. These rules define a serialization mechanism that converts the application data types to XML and XML to data types.
- *remote procedure call* definitions

SOAP 1.1 specification is a W3C note. (The W3C XML Protocol Working Group has been formed to create a standard that will supersede SOAP.)

SOAP is transport protocol-independent and operating system-independent. It provides the standard XML message format for all applications. SOAP uses the W3C XML Schema standard of the World Wide Web Consortium (W3C).

See Also:

- `http://www.w3.org/TR/SOAP/`
- http://xml.apache.org/soap

A SOAP service remote procedure call (RPC) request and response sequence includes the steps:

1. A SOAP client writes a request for service in a conforming XML document, using either an editor or the Oracle SOAP client API.
2. The client sends the document to a SOAP Request Handler running as a servlet on a Web server.
3. The Web Server dispatches the message as a service request to an appropriate server-side application providing the requested service.
4. The application must verify that the message contains supported parts. The response from the service is returned to the SOAP Request Handler servlet and then to the caller using the SOAP payload format.

What Are UDDI and WSDL?

The Universal Description, Discovery and Integration (UDDI) specification provides a platform-independent framework using XML to describe services, discover businesses, and integrate business services on the Internet. The UDDI business registry is the public database where companies are registered. The UDDI business registration is an XML file with three sections:

- *white pages* that include address, contact, and known identifiers
- *yellow pages* include industrial categorization
- *green pages* containing the technical information about exposed services

The Web Services Description Language (WSDL) is a general purpose XML language for describing the interface, protocol bindings, and deployment details of Web services. WSDL provides a method of describing the abstract interface and arbitrary network services. A WSDL service is registered or embedded in the UDDI registry.

The stack of protocols used in Web services is summarized in the following table:

Protocol Stack
Universal Service Interoperability Protocols (WSDL, and so on.)
Universal Description, Discovery Integration (UDDI)
Simple Object Access Protocol (SOAP)
XML, XML Schema
Internet Protocols (HTTP, HTTPS, TCP / IP)

What Is Oracle SOAP?

Oracle SOAP is an implementation of the Simple Object Access Protocol. Oracle SOAP is based on the SOAP open source implementation developed by the Apache Software Foundation.

How Does SOAP Work?

Consider this example: a GetLastTradePrice SOAP request is sent to a StockQuote service. The request takes a string parameter, the company stock symbol, and returns a float in the SOAP response. The XML document represents the SOAP message. The SOAP envelope element is the top element of the XML document. XML namespaces are used to clarify SOAP identifiers from application-specific identifiers. The following example uses HTTP as the transport protocol. The rules governing XML payload format in SOAP are independent of the fact that the payload is carried in HTTP. The SOAP request message embedded in the HTTP request is:

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"
<SOAP-ENV:Envelope  xmlns:SOAP-   ENV="http://schemas.xmlsoap.org/soap/
envelope/"  SOAP-
ENV:encodingStyle="http://schemas.xnlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<m:GetLastTradePrice xmlns:m="Some-URI">
<symbol>ORCL</symbol>
<m:GetLastTradePrice>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Here is the response HTTP message:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope  xmlns:SOAP-
ENV=http://schemas.xmlsoap.org/soap//envelope/ SOAP-
ENV:encodingStyle="http://schemas.xnlsoap.org/soap/encoding/"/>
<SOAP-ENV:Body>
<m:GetLastTradePriceResponse xmlns:m="Some-URI">
<Price>34.5</Price>
```

```
</m:GetLastTradePriceResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

What Is a SOAP Client?

A SOAP client application represents a user-written application that makes SOAP requests. The SOAP client has these capabilities:

- Gathers all parameters that are needed to invoke a service.
- Creates a SOAP service request message. This is an XML message that is built according to the SOAP protocol and that contains all the values of all input parameters encoded in XML. This process is called *serialization*.
- Submits the request to a SOAP server using some transport protocol that is supported by the SOAP server.
- Receives a SOAP response message.
- Determines the success or failure of the request by handling the SOAP Fault element.
- Converts the returned parameter from XML to native data type. This process is called *deserialization*.
- Uses the result as needed.

SOAP Client API

SOAP clients generate the XML documents that compose a request for a SOAP service and handle the SOAP response. Oracle SOAP processes requests from any client that sends a valid SOAP request. To facilitate client development, Oracle SOAP includes a SOAP client API that provides a generic way to invoke a SOAP service.

The SOAP client API supports a synchronous invocation model for requests and responses. The SOAP client API makes it easier for you to write a Java client application to make a SOAP request. The SOAP client API encapsulates the creation of the SOAP request and the details of sending the request over the underlying transport protocol. The SOAP client API also supports a pluggable transport, allowing the client to easily change the transport (available transports include HTTP and HTTPS).

What Is a SOAP Server?

A SOAP server has the following capabilities:

- The server receives the service request.
- The server parses the XML request and then decides to execute the message or reject it.
- If the message is executed, the server determines if the requested service exists.
- The server converts all input parameters from XML into data types that the service understands.
- The server invokes the service.
- The return parameter is converted to XML and a SOAP response message is generated.
- The response message is sent back to the caller.

Oracle SOAP Security Features

Oracle SOAP uses the security capabilities in the transport to support secure access and to support other security features. For example, using HTTPS, Oracle SOAP provides confidentiality, authentication, and integrity over the Secure Sockets Layer (SSL). Other security features such as logging and authorization, are provided by the service provider.

SOAP Transports

SOAP transports are the protocols that carry SOAP messages. Oracle SOAP supports the following transports:

- HTTP: This protocol is the basic SOAP transport. The Oracle SOAP Request Handler Servlet manages HTTP requests and supplies responses directly over HTTP.This protocol is becoming a standard because of its popularity.
- HTTPS: The Oracle SOAP Request Handler Servlet manages HTTPS requests and supplies responses, with different security levels supported.

Administrative Clients

SOAP administrative clients include the Service Manager and the Provider Manager. These administrative clients are services that support dynamic deployment of new services and new providers.

SOAP Request Handler

The SOAP Request Handler is a Java servlet that receives SOAP requests, looks up the appropriate service provider, handles the service provider that invokes the requested method (service), and returns the SOAP response, if any.

SOAP Provider Interface and Providers

Oracle SOAP includes a provider implementation for Java classes. Other providers can be added.

Provider Interface

The provider interface allows the SOAP server to uniformly invoke service methods regardless of the type of provider (Java class, stored procedure, or some other provider type). There is one provider interface implementation for each type of service provider, and it encapsulates all provider-specific information. The provider interface makes SOAP implementation easily extensible to support new types of service providers.

Provider Deployment Administration

Oracle SOAP provides the provider deployment administration client to manage provider deployment information.

SOAP Services

SOAP application developers provide SOAP services. These services are made available using the supplied default Java class provider or custom providers. Oracle SOAP includes a service deployment administration client that runs as a service to manage services. SOAP services, including Java services, represent user-written applications that are provided to remote SOAP clients.

JDeveloper Support for SOAP

Oracle9*i* JDeveloper has WSDL, SOAP, and UDDI support.

> **See Also:** Chapter 24, "Developing XML Applications with JDeveloper"

See the Developer's Guides

Here's how to find the Oracle9iAS SOAP Developer's Guide, Release 1 (v1.0.2.2), May 2001, PN A90297-01 online:

1. Open `http://otn.oracle.com/docs/products/ias/content.html`
2. Open the Generic Documentation Library for 1.0.2.2.x.
3. Click on the Integrate Users, Applications, and Businesses link.

See Also: For more information about Oracle SOAP and Web Services, including documentation and downloads, see:

- http://otn.oracle.com/products/ias/daily/sept07.html
- *Oracle9i Application Developer's Guide - Advanced Queuing* for a discussion of Internet access to AQ (Advanced Queuing.
- *Oracle9i XML API Reference - XDK and Oracle XML DB*
- The SOAP API is on the Product CD, Disk 1, in file `doc/readmes/ADDEN_rdbms.htm`

12

Oracle TransX Utility

This chapter contains the following sections:

Overview of the TransX Utility

The TransX Utility simplifies the loading of translated seed data and messages into a database. It also reduces internationalization costs by:

- Preparing strings to be translated.
- Translating the strings.
- Loading the strings to the database.

The TransX Utility minimizes translation data format errors and it accurately loads the translation contents into pre-determined locations in the database. Other advantages of the TransX Utility are:

- Translation vendors no longer have to work with unfamiliar SQL and PL/SQL scripts.
- Syntax errors due to varying Globalization Support settings are eliminated.
- The UNISTR construct is no longer required for every piece of NCHAR data.

Development groups that need to load translated messages and seed data can use the TransX Utility to simplify what it takes for meeting internationalization requirements. Once the data is in a predefined format, the TransX Utility validates its format.

Choosing the correct encoding when loading translated data is automated because loading with TransX takes advantage of XML which describes the encoding. This means that loading errors due to incorrect encoding is impossible as long as the data file conforms to the XML standard.

Primary TransX Utility Features

This section describes the following features of the TransX Utility:

- Simplified Multilingual Data Loading
- Simplified Data Format Support and Interface
- Loading Dataset in The Standard XML Format
- Handling Existing Data
- Other TransX Utility Features

Simplified Multilingual Data Loading

Traditionally, the typical translation data loading method was to switch the NLS_LANG setting when you switch files to be loaded. Each of the load files is encoded in a particular character set suitable for the particular language. This was required because translations must be done in the same file format (typically in .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. The TransX Utility loading tool frees the development and translation groups maintaining the correct character set throughout the process until they successfully load the data into the database using XML.

Simplified Data Format Support and Interface

The TransX Utility data loading tool complies with a data format defined to be the canonical method for the representation of any seed data to be loaded to the database. The format is intuitive and easy to understand. The format is also simplified for translation groups to use. The format specification defines how translators can describe the data to load it in the expected way.

The data loading tool has a command-line interface and programmable API. Both of them are straightforward and require little time to learn.

Loading Dataset in The Standard XML Format

Given the dataset in the canonical format, the TransX Utility loads the data into the designated locations in the database. It does not, however, create objects, including the table that the data is going to be loaded to. In addition to literal values represented in XML, the following expressions can be used to describe the data to be loaded:

Constant Expression A constant expression allows you to specify a constant value. A column with a fixed value for each row does not have to repeat the same value.

Sequence A column can be loaded with a value obtained from a sequence in the database.

Query A SQL query can be used to load a column. A query can use parameter(s).

Handling Existing Data

The data loading tool determines whether there are duplicate rows in the database. It also lets you choose how it processes duplicate rows from one of the options in

the following list. A row is considered duplicate if the values of all columns specified as lookup-key are the same. The processing options are:

- Skip the duplicate rows or leave them as they are (default)
- Update or overwrite the duplicate rows with the data in provided dataset
- Display an error

Other TransX Utility Features

The lists describes other TransX Utility features:

- Command-line Interface—The data loading tool provides easy-to-use commands.
- User API—The data loading tool exposes a Java API.
- Validation—The data loading tool validates the data format and reports errors.
- White Space Handling—White space characters in the dataset are not significant, unless otherwise specified in various granularity.
- Unloading—Based on a query, the data loading tool exports the result into the standard data format.
- Intimacy with Translation Exchange Format—Designed for transformation to and from translation exchange format
- Localized User Interface—Messages are provided in *N* languages.

Installing TransX Utility

Here is how to install TransX, and the dependencies of TransX.

Dependencies of TransX

The Oracle TransX utility needs the following components in order to function:

- Database connectivity -- JDBC drivers. The utility can work with any JDBC drivers but is optimized for Oracle's JDBC drivers. Oracle does not guarantee or provide support for TransX running against non-Oracle databases.
- XML Parser -- Oracle XML Parser, Version 2. The Oracle XML Parser, Version 2, is part of the Oracle8*i* and Oracle9*i* installations, and is also available from the Oracle Technology Network (OTN) Web site.

- XML Schema Processor -- Oracle XML Schema Processor. The Oracle XML Schema Processor is part of the Oracle8*i* and Oracle9*i* installations, downloadable from the Oracle Technology Network (OTN) Web site.
- XML SQL Utility-- Oracle XML SQL Utility (XSU). The Oracle XSU is part of the Oracle8*i* and Oracle9*i* installation, and is also available from Oracle Technology Network (OTN) Web site.

Installing TransX Using the Oracle Installer

TransX is packaged with Oracle9*i*. The TransX utility is made up of three executable files:

- $ORACLE_HOME/rdbms/jlib/transx.zip -- contains all the java classes which make up TransX $ORACLE_HOME/rdbms/bin/transx -- a shell script to invoke TransX from UNIX command line.
- $ORACLE_HOME/rdbms/bin/transx.bat -- a batch file to invoke TransX from Windows command line.

By default, the Oracle9*i* installer installs TransX on your hard drive in the locations specified above.

Installing TransX Downloaded from OTN

Download the correct XDK for java distribution archive from the Oracle Technology Network (http://otn.oracle.com). Expand the downloaded archive. Depending on the usage scenario, perform the following install tasks:

To use the TransX's front-end or its Java API, you need to:

Set up the environment (that is, set CLASSPATH) using the `env.xxx` script (located in the bin directory inside the directory created by extracting the XDK download archive):

Unix users: make sure that the path names in env.csh are correct; source the env.csh. If you are using a shell other than `csh` or `tcsh`, you will have to edit the file to use your shell's syntax.

Windows users: make sure that the path names in `env.bat` are correct; execute the file.

TransX Utility Command-Line Syntax

The following describes the command-line syntax for the TransX Utility.

```
java oracle.xml.transx.loader [options] connect_string username password
datasource [datasource]
java oracle.xml.transx.loader -v datasource [datasource]
java oracle.xml.transx.loader -x connect_string username password table [column]
java oracle.xml.transx.loader -s connect_string username password filename table
[column]
```

TransX Utility Command-Line Examples

The following are command-line examples for the TransX Utility:

```
java oracle.xml.transx.loader "dlsun9999:1521:mydb" scott tiger foo.xml
java oracle.xml.transx.loader "jdbc:oracle:oci:@mydb" scott tiger foo.xml
java oracle.xml.transx.loader -v foo.xml
java oracle.xml.transx.loader -x "dlsun9999:1521:mydb" scott tiger emp
java oracle.xml.transx.loader -s "dlsun9999:1521:mydb" scott tiger emp.xml emp
ename job
```

TransX Utility Command-line Parameters

Table 12–1 shows the command-line parameters.

Table 12–1 TransX Utility Command-line Parameters

Parameter	Meaning
connect_string	JDBC connect string You can omit the connect string information through the '@' symbol. 'jdbc:oracle:thin:@' will be supplied.
username	Database user name.
password	Password for the database user.
datasource	An XML data source.
option	Options in Table 12–2, "TransX Utility Command-line Options".

TransX Utility Command-line Options

Table 12–2 TransX Utility Command-line Options

Option	Meaning	Description
-u	Update existing rows.	When this option is specified, existing rows are not skipped but updated. To exclude a column from the update operation, specify the `useforupdate` attribute to be "no".
-e	Raise exception if a row is already existing in the database.	When this option is specified, an exception will be thrown if a duplicate row is found. By default, duplicate rows are simply skipped. Rows are considered duplicate if the values for lookup-key column(s) in the database and the dataset are the same.
-x	Print data in the database in the predefined format*.	Similar to the -s option, it causes TransX to perform the opposite operation of loading. Unlike the -s option, it prints the output to `stdout`. Note: Redirecting this output to a file is discouraged, because intervention of the operating system may result in data loss due to unexpected transcoding.
-s	Save data in the database into a file in the predefined format*.	This is an option to perform unloading. It queries the database, formats the result into the predefined XML format and store it under the specified file name.
-p	Print the XML to load.	Prints out the dataset for insert in the canonical format of XSU.
-t	Print the XML for update.	Prints out the dataset for update in the canonical format of XSU.
-o	Omit validation (as the dataset is parsed it is validated by default).	Causes TransX to skip the format validation, which is performed by default.
-v	Validate the data format and exit without loading.	Causes TransX to perform validation and exit.
-w	Preserve white space.	Causes TransX to treat whitespace characters (such as \t, \r, \n, and ' ') as significant. Consecutive whitespace characters in string data elements are condensed into one space character by default.

Command-line Option Exceptions The following are the command-line option exceptions:

- -u and -e are mutually exclusive
- -v must be the only option followed by data, as in the examples
- -x must be the only option followed by connect info and SQL query as in the examples

Omitting all arguments will result in the display of the front-end usage information shown in the table.

For complete details of the Java API for TransX Utility:

See Also: *Oracle9i XML API Reference - XDK and Oracle XML DB*

Sample Code for TransX Utility

The following is sample code for the TransX Utility:

```
String  datasrc[] = {"data1.xml", "data2.xml", "data3.xml"};

// instantiate a loader
TransX  transx = loader.getLoader();

// start a data loading session
transx.open( jdbc_con_str, usr, pwd );

// specify operation modes
transx.setLoadingMode( LoadingMode.SKIP_DUPLICATES );
transx.setValidationMode( false );

// load the dataset(s)
for ( int i = 0 ; i < datasrc.length ; i++ )
{
transx.load( datasrc[i] );
}

// cleanup
transx.close();
```

Part III

XDK for C/C++

These chapters describes how to access and use XML Developer's Kit (XDK) for C/C++:

13

XML Parser for C

This chapter contains the following sections:

Accessing XML Parser for C

The XML Parser for C is provided with Oracle9*i* and Oracle9*i* Application Server. It is also available for download from the OTN site:
`http://otn.oracle.com/tech/xml`

It is located in `$ORACLE_HOME/xdk/c/parser` on Solaris™ Operating Environment systems.

XML Parser for C Features

`readme.html` in the root directory of the software archive contains release specific information including bug fixes and API additions.

XML Parser for C will check if an XML document is well-formed, and optionally validate it against a DTD. The parser constructs an object tree which can be accessed through a DOM interface or operate serially through a SAX interface.

You can post questions, comments, or bug reports to the XML Discussion Forum at `http://otn.oracle.com/tech/xml`.

Specifications

See Also:

- The `doc` directory in your install area
- *Oracle9i XML API Reference - XDK and Oracle XML DB*
- `http://otn.oracle.com/tech/xml/`

Memory Allocation

The memory callback functions `memcb` may be used if you wish to use your own memory allocation. If they are used, all of the functions should be specified.

The memory allocated for parameters passed to the SAX callbacks or for nodes and data stored with the DOM parse tree will not be freed until one of the following is done:

- `xmlclean()` is called.
- `xmlterm()` is called.

Thread Safety

If threads are forked off somewhere in the midst of the init-parse-term sequence of calls, you will get unpredictable behavior and results.

Data Types Index

Table 13–1 lists the datatypes used in XML Parser for C.

Table 13–1 Datatypes Used in XML Parser for C

DataType	Description
oratext	String pointer
xmlctx	Master XML context
xmlmemcb	Memory callback structure (optional)
xmlsaxcb	SAX callback structure (SAX only)
ub4	32-bit (or larger) unsigned integer
uword	Native unsigned integer

Error Message Files

Error message files are provided in the mesg/ subdirectory. The messages files also exist in the `$ORACLE_HOME/xdk/mesg` directory. You may set the environment variable `ORA_XML_MESG` to point to the absolute path of the `mesg/` subdirectory although this not required.

Validation Modes

See Also: Available validation modes are described in "Oracle XML Parsers Validation Modes" on page 4-5.

XML Parser for C Usage

Figure 13–1 describes XML Parser for C calling sequence as follows:

1. `xmlinit()` function initializes the parsing process.
2. The parsed item can be an XML document (file) or string buffer. If the input is an XML document or file, it is parsed using the `xmlparser()` function. If the input is a string buffer, it is parsed using the `xmlparserbuf()` function.

3. DOM or SAX API:

   DOM: If you are using the DOM interface, include the following steps:

   - The `xmlparse()` or `xmlparseBuffer()` function calls `.getDocumentElement()`. If no other DOM functions are being applied, you can invoke `xmlterm()`.
   - This optionally calls other DOM functions if required. These are typically Node or print functions. It outputs the DOM document.
   - If complete, the process invokes `xmlterm()`
   - You can first invoke `xmlclean()` to clean up any data structures created during the parse process. You would then call `xmlterm()`

   SAX: If you are using the SAX interface, include the following steps:

   - Process the results of the parser from `xmlparse()` or `xmlparseBuf()` using callback functions.
   - Register the callback functions.

4. Use `xmlclean()` to clean up the memory and structures used during a parse, and go to Step 5. or return to Step 2.
5. Terminate the parsing process with `xmlterm()`

XML Parser for C usage is further explained in Figure 13–1.

Parser Calling Sequence

The sequence of calls to the parser can be any of the following:

- `xmlinit() - xmlparse()` or

  `xmlparsebuf() - xmlterm()`
- `xmlinit() - xmlparse()` or

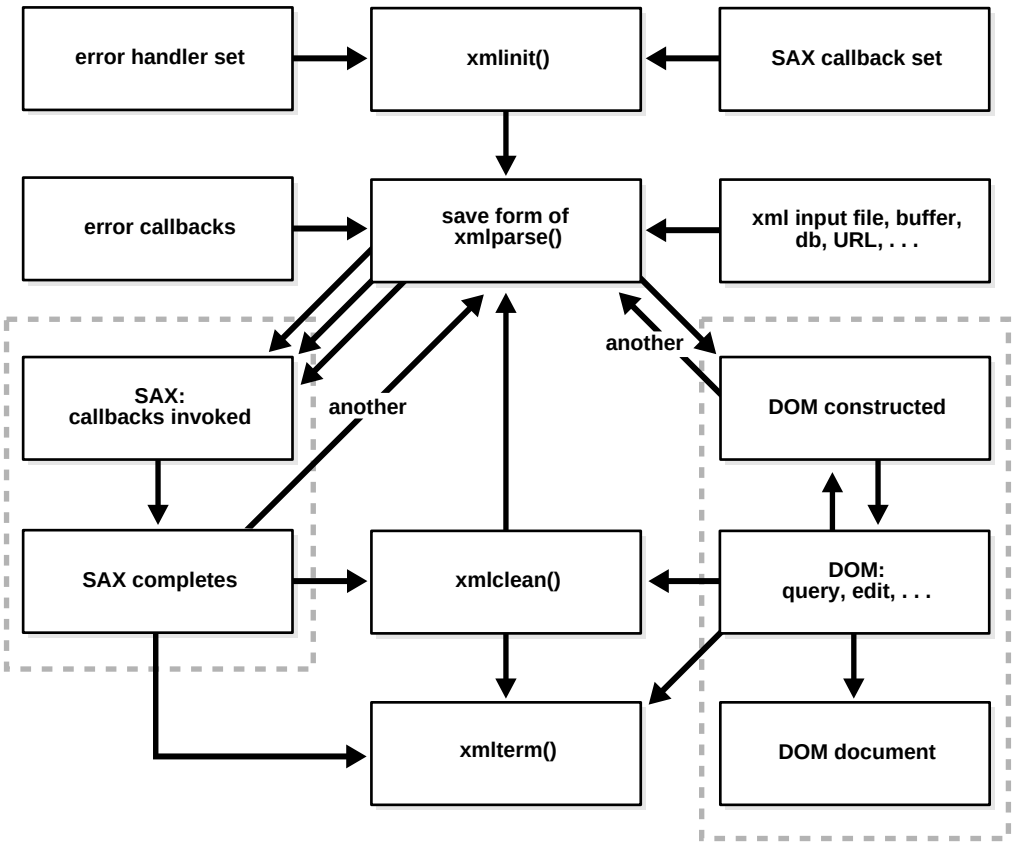
  `xmlparsebuf() - xmlclean() - xmlparse()` or

  `xmlparsebuf() - xmlclean() -... - xmlterm()`
- `xmlinit() - xmlparse()` or

  `xmlparsebuf() - xmlparse()` or

  `xmlparsebuf() -... - xmlterm()`

Figure 13–1 XML Parser for C Calling Sequence



XML Parser for C Default Behavior

The following is the XML Parser for C default behavior:

- Character set encoding is UTF-8. If all your documents are ASCII, you are encouraged to set the encoding to US-ASCII for better performance.
- Messages are printed to stderr unless msghdlr is given.

- A parse tree which can be accessed by DOM APIs is built unless saxcb is set to use the SAX callback APIs. Note that any of the SAX callback functions can be set to NULL if not needed.
- The default behavior for the parser is to check that the input is well-formed but not to check whether it is valid. The flag XML_FLAG_VALIDATE can be set to validate the input. The default behavior for whitespace processing is to be fully conformant to the XML 1.0 spec, that is, all whitespace is reported back to the application but it is indicated which whitespace is ignorable. However, some applications may prefer to set the XML_FLAG_DISCARD_WHITESPACE which will discard all whitespace between an end-element tag and the following start-element tag.

Note: It is recommended that you set the default encoding explicitly if using only single byte character sets (such as US-ASCII or any of the ISO-8859 character sets) for performance up to 25% faster than with multibyte character sets, such as UTF-8.

DOM and SAX APIs

Oracle XML parser for C checks if an XML document is well-formed, and optionally validates it against a DTD. The parser constructs an object tree which can be accessed through one of the following interfaces:

- DOM interface
- Serially through a SAX interface

These two XML APIs:

- DOM: Tree-based APIs. A tree-based API compiles an XML document into an internal tree structure, then allows an application to navigate that tree using the Document Object Model (DOM), a standard tree-based API for XML and HTML documents.
- SAX: Event-based APIs. An event-based API, on the other hand, reports parsing events (such as the start and end of elements) directly to the application through callbacks, and does not usually build an internal tree. The application implements handlers to deal with the different events, much like handling events in a graphical user interface.

Tree-based APIs are useful for a wide range of applications, but they often put a great strain on system resources, especially if the document is large (under very

controlled circumstances, it is possible to construct the tree in a lazy fashion to avoid some of this problem). Furthermore, some applications need to build their own, different data trees, and it is very inefficient to build a tree of parse nodes, only to map it onto a new tree.

In both of these cases, an event-based API provides a simpler, lower-level access to an XML document: you can parse documents much larger than your available system memory, and you can construct your own data structures using your callback event handlers.

Using the SAX API

To use SAX, an `xmlsaxcb` structure is initialized with function pointers and passed to the `xmlinit()` call. A pointer to a user-defined context structure can also be included. That context pointer will be passed to each SAX function.

SAX Callback Structure

The SAX callback structure:

```
typedef struct
{
 sword (*startDocument)(void *ctx);
 sword (*endDocument)(void *ctx);
 sword (*startElement)(void *ctx, const oratext *name,
            const struct xmlarray *attrs);
 sword (*endElement)(void *ctx, const oratext *name);
 sword (*characters)(void *ctx, const oratext *ch, size_t len);
 sword (*ignorableWhitespace)(void *ctx, const oratext *ch, size_t len);
 sword (*processingInstruction)(void *ctx, const oratext *target,
            const oratext *data);
 sword (*notationDecl)(void *ctx, const oratext *name,
            const oratext *publicId, const oratext *systemId);
 sword (*unparsedEntityDecl)(void *ctx, const oratext *name,
            const oratext *publicId,
            const oratext *systemId, const oratext *notationName);
 sword (*nsStartElement)(void *ctx, const oratext *qname,
            const oratext *local, const oratext *nsp,
            const struct xmlnodes *attrs);
} xmlsaxcb;
```

Invoking XML Parser for C

XML Parser for C can be invoked in two ways:

- By invoking the executable on the command line
- By writing C code and using the supplied APIs

Command Line Usage

The XML Parser for C can be called as an executable by invoking `bin/xml`

Table 13–2 lists the command line options.

Table 13–2 XML Parser for C: Command Line Options

Option	Description
-c	Conformance check only, no validation
-e encoding	Specify input file encoding
-h	Help - show this usage help
-n	Number - DOM traverse and report number of elements
-p	Print document and DTD structures after parse
-x	Exercise SAX interface and print document
-v	Version - display parser version then exit
-w	Whitespace - preserve all whitespace

Writing C Code to Use Supplied APIs

XML Parser for C can also be invoked by writing code to use the supplied APIs. The code must be compiled using the headers in the include/ subdirectory and linked against the libraries in the lib/ subdirectory. Please see the `Makefile` in the sample/ subdirectory for full details of how to build your program.

Using the Sample Files Included with Your Software

$ORACLE_HOME/xdk/c/parser/`sample/` directory contains several XML applications to illustrate how to use the XML Parser for C with the DOM and SAX interfaces.

Table 13–3 lists the sample files in sample/ directory.

Table 13–3 XML Parser for C sample/ Files

sample/ File Name	Description
DOMNamespace.c	Source for DOMNamespace program
DOMNamespace.std	Expected output from DOMNamespace
DOMSample.c	Source for DOMSample program
DOMSample.std	Expected output from DOMSample
FullDOM.c	Sample usage of DOM interface
FullDOM.std	Expected output from FullDOM
Make.bat	Batch file for building sample programs
NSExample.xml	Sample XML file using namespaces
SAXNamespace.c	Source for SAXNamespace program
SAXNamespace.std	Expected output from SAXNamespace
SAXSample.c	Source for SAXSample program
SAXSample.std	Expected output from SAXSample
XSLSample.c	Source for XSLSample program
XSLSample.std	Expected output from XSLSample
class.xml	XML file that may be used with XSLSample
iden.xsl	Stylesheet that may be used with XSLSample
cleo.xml	The Tragedy of Antony and Cleopatra XML version of Shakespeare's play

Running the XML Parser for C Sample Programs

Building the Sample Programs

Change directories to the sample directory (`$ORACLE_HOME/xdk/demo/c/parser` on Solaris™ Operating Environment) and read the README file. This will explain how to build the sample programs according to your platform.

Sample Programs

Table 13–4 lists the programs built by the sample files in the sample directory.

Table 13–4 XML Parser for C: Sample Built Programs in sample/

Built Program	Description
DOMSample	A sample application using DOM APIs (shows an outline of Cleopatra, that is, the XML elements ACT and SCENE).
SAXSample [word]	A sample application using SAX APIs. Given a word, shows all lines in the play Cleopatra containing that word. If no word is specified, 'death' is used.
DOMNamespace	Same as SAXNamespace except using DOM interface.
SAXNamespace	A sample application using Namespace extensions to SAX API; prints out all elements and attributes of NSExample.xml along with full namespace information.
FullDOM	Sample usage of full DOM interface. Exercises all the calls, but does nothing too exciting.
XSLSample <xmlfile> <xsl ss>	Sample usage of XSL processor. It takes two filenames as input, the XML file and XSL stylesheet

14

XSLT Processor for C

This chapter contains the following sections:

Accessing XSLT for C

XSLT for C is provided with Oracle9*i* and Oracle9*i* Application Server. It is also available for download from the OTN site:
`http://otn.oracle.com/tech/xml`

It is located in `$ORACLE_HOME/xdk/c/parser`.

XSLT for C Features

`readme.html` in the root directory of the software archive contains release specific information including bug fixes and API additions.

You can post questions, comments, or bug reports to the XML Discussion Forum at `http://otn.oracle.com/tech/xml`.

Specifications

See the following:

See Also:

- The doc directory in your install area
- *Oracle9i XML API Reference - XDK and Oracle XML DB*
- `http://otn.oracle.com/tech/xml/`

XML XSLT for C (DOM Interface) Usage

Figure 14–1 shows the XSLT for C functionality.

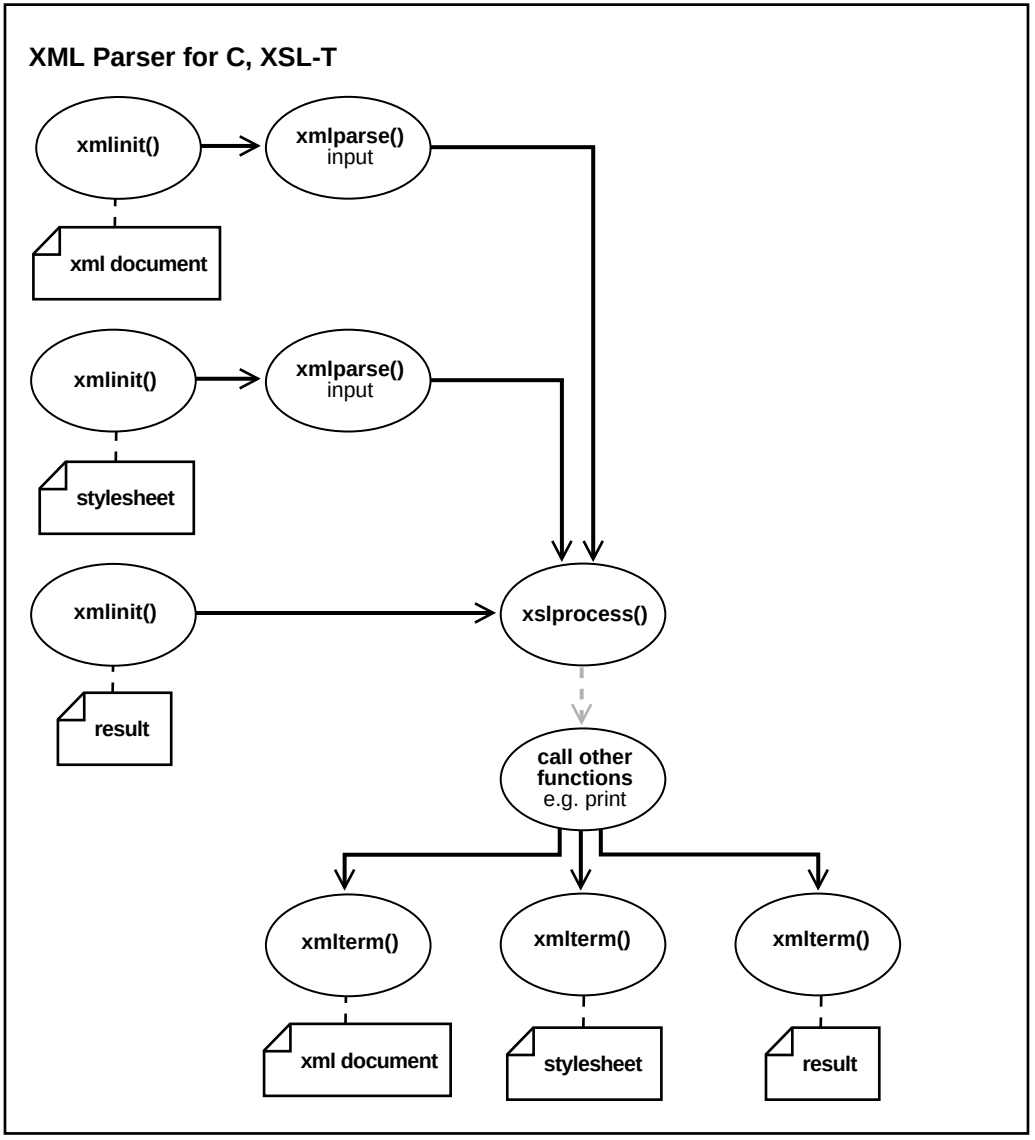
1. There are two inputs to `xmlparse()`:
   - The stylesheet to be applied to the XML document
   - XML document
2. `xmlinit()` initializes the XSLT processing. `xmlinit()` initializes the `xslprocess()` result.
3. `xslprocess()`optionally calls other functions, such as print functions. You can see the list of available functions either on OTN or in the *Oracle9i XML API Reference - XDK and Oracle XML DB*.
4. The resultant document (XML, HTML, VML, and so on) is typically sent to an application for further processing.

5. The application terminates the XSLT process by declaring `xmlterm()` for the XML document, stylesheet, and final result.

XML Parser for C's XSLT functionality is illustrated with the following examples:

- XSLT for C Example 2: C — XSLSample.c on page 14-6
- XSLT for C Example 3: C — XSLSample.std on page 14-9

Figure 14–1 XSLT for C (DOM Interface) Usage



Invoking XSLT for C

XSLT for C can be invoked in two ways:

- By invoking the executable on the command line
- By writing C code and using the supplied APIs

Command Line Usage

The XSLT for C can be called as an executable by invoking `bin/xml`

Table 14–1 lists the command line options.

Table 14–1 XML Parser for C: Command Line Options

Option	Description
-e encoding	Specify input file encoding
-h	Help - show this usage help
-v	Version - display parser version then exit
-w	Whitespace - preserve all whitespace
-s	Stylesheet

Using the Sample Files Included with the Software

`$ORACLE_HOME/xdk/c/parser/sample` directory contains several XML applications to illustrate how to use the XSLT for C.

Table 14–2 lists the sample files in sample/ directory.

Table 14–2 XSLT for C sample/ Files

sample/ File Name	Description
XSLSample.c	Source for XSLSample program
XSLSample.std	Expected output from XSLSample
class.xml	XML file that may be used with XSLSample
iden.xsl	Stylesheet that may be used with XSLSample
cleo.xml	XML version of Shakespeare's play

Running the XSLT for C Sample Programs

Building the Sample Programs

Change directories to the sample directory and read the README file. This will explain how to build the sample programs according to your platform.

Sample Programs

Table 14–3 lists the programs built by the sample files in the sample directory.

Table 14–3 XSLT for C: Sample Built Programs in sample/

Built Program	Description
XSLSample <xmlfile> <xsl ss>	Sample usage of XSL processor. It takes two filenames as input, the XML file and XSL stylesheet

XSLT for C Example1: XSL — iden.xsl

This example stylesheet can be used to input `XSLSample.c`.

```
<?xml version="1.0"?>
<!-- Identity transformation -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="*|@*|comment()|processing-instruction()|text()">
      <xsl:copy>
          <xsl:apply-templates
select="*|@*|comment()|processing-instruction()|text()"/>
      </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

XSLT for C Example 2: C — XSLSample.c

This example contains C source code for `XSLSample.c`.

```
/* Copyright (c) Oracle Corporation 1999. All Rights Reserved. */

/*
   NAME
     XSLSample.c - Sample function for XSL
   DESCRIPTION
```

```
     Sample usage of C XSL Processor
*/

#include <stdio.h>
#ifndef ORATYPES
# include <oratypes.h>
#endif

#ifndef ORAXML_ORACLE
# include <oraxml.h>
#endif

int main(int argc, char *argv[])
{
    xmlctx     *xctx, *xslctx, *resctx;
    xmlnode    *result;
    uword       ecode;
    /* Check for correct usage */
    if (argc < 3)
        {
        puts("Usage is XSLSample <xmlfile> <xslfile>\n");
        return 1;
        }

    /* Parse the XML document */
    if (!(xctx = xmlinit(&ecode, (const oratext *) 0,
                         (void (*)(void *, const oratext *, uword)) 0,
                         (void *) 0, (const xmlsaxcb *) 0, (void *) 0,
                         (const xmlmemcb *) 0, (void *) 0,
                         (const oratext *) 0)))
    {
        printf("Failed to initialze XML parser, error %u\n", (unsigned) ecode);
        return 1;
    }

    printf("Parsing '%s' ...\n", argv[1]);
    if (ecode = xmlparse(xctx, (oratext *)argv[1], (oratext *) 0,
                         XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE))
    {
        printf("Parse failed, error %u\n", (unsigned) ecode);
        return 1;
    }

    /* Parse the XSL document */
    if (!(xslctx = xmlinit(&ecode, (const oratext *) 0,
```

```
                    (void (*)(void *, const oratext *, uword)) 0,
                    (void *) 0, (const xmlsaxcb *) 0, (void *) 0,
                    (const xmlmemcb *) 0, (void *) 0,
                    (const oratext *) 0)))
{
    printf("Failed to initialze XML parser, error %u\n", (unsigned) ecode);
    return 1;
}

printf("Parsing '%s' ...\n", argv[2]);
if (ecode = xmlparse(xslctx, (oratext *)argv[2], (oratext *) 0,
                    XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE))
{
    printf("Parse failed, error %u\n", (unsigned) ecode);
    return 1;
}

/* Initialize the result context */
if (!(resctx = xmlinit(&ecode, (const oratext *) 0,
                    (void (*)(void *, const oratext *, uword)) 0,
                    (void *) 0, (const xmlsaxcb *) 0, (void *) 0,
                    (const xmlmemcb *) 0, (void *) 0,
                    (const oratext *) 0)))
{
    printf("Failed to initialze XML parser, error %u\n", (unsigned) ecode);
    return 1;
}

/* XSL processing */
printf("XSL Processing\n");
if (ecode = xslprocess(xctx, xslctx, resctx, &result))
{
    printf("Parse failed, error %u\n", (unsigned) ecode);
    return 1;
}

/* Print the result tree */
printres(resctx, result);

/* Call the terminate functions */
(void)xmlterm(xctx);
(void)xmlterm(xslctx);
(void)xmlterm(resctx);

return 0;
```

```
}
```

XSLT for C Example 3: C — XSLSample.std

`XSLSample.std` shows the expected output from `XSLSample.c`.

```
Parsing 'class.xml' ...
Parsing 'iden.xsl' ...
XSL Processing
<root>
   <course>
      <Name>Calculus</Name>
      <Dept>Math</Dept>
      <Instructor>
         <Name>Jim Green</Name>
      </Instructor>
      <Student>
         <Name>Jack</Name>
         <Name>Mary</Name>
         <Name>Paul</Name>
      </Student>
   </course>
</root>
```

15

XML Schema Processor for C

This chapter contains the following sections:

Oracle XML Schema Processor for C

The XML Schema Processor for C is a companion component to the XML Parser for C. It allows support for simple and complex datatypes in Oracle9*i* XML applications.

The XML Schema Processor for C supports the W3C XML Schema Recommendation, with the goal being that it be 100% fully conformant when XML Schema becomes a W3C Recommendation. This makes writing custom applications that process XML documents straightforward in the Oracle9i environment, and means that a standards-compliant XML Schema Processor is part of the Oracle9i platform on every operating system where Oracle9i is ported.

> **See Also:** Chapter 4, "XML Parser for Java", for more information about XML Schema and why you would want to use XML Schema.

Oracle XML Schema for C Features

XML Schema Processor for C has the following features:

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

> **See Also:**
>
> - *Oracle9i XML API Reference - XDK and Oracle XML DB*

Online Documentation

Documentation for Oracle XML Schema Processor for C is located in the doc directory in your install area.

Standards Conformance

The Schema Processor conforms to the following standards:

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

XML Schema Processor for C: Supplied Software

Table 15–1 lists the supplied files and directories with this release:

Table 15–1 XML Schema Processor for C: Supplied Files

Directory an d Files	Description
license.html	Licensing agreement
readme.html	This file
bin	Schema processor executable, “schema”
doc	API documentation
include	header files
lib	XML / XSL / Schema & support libraries
mesg	Error message files
sample	Example usage of the Schema processor

Table 15–2 lists the included libraries:

Table 15–2 XML Schema Processor for C: Supplied Libraries

Included Library	Description
libxml9.a	XML Parser / XSL Processor
libxsd9.a	XML Schema Processor
libcore9.a	CORE functions
libnls9.a	National Language Support

Invoking XML Schema Processor for C

XML Schema Processor for C can be called as an executable by invoking `bin/schema` in the install area. This takes two arguments:

- XML instance document
- Optionally, a default schema

The Schema Processor can also be invoked by writing code using the supplied APIs. The code must be compiled using the headers in the include / subdirectory and linked against the libraries in the lib / subdirectory. See Makefile in the sample / subdirectory for details on how to build your program.

An error message file is provided in the mesg/ subdirectory. Currently, the only message file is in English although message files for other languages may be supplied in future releases.

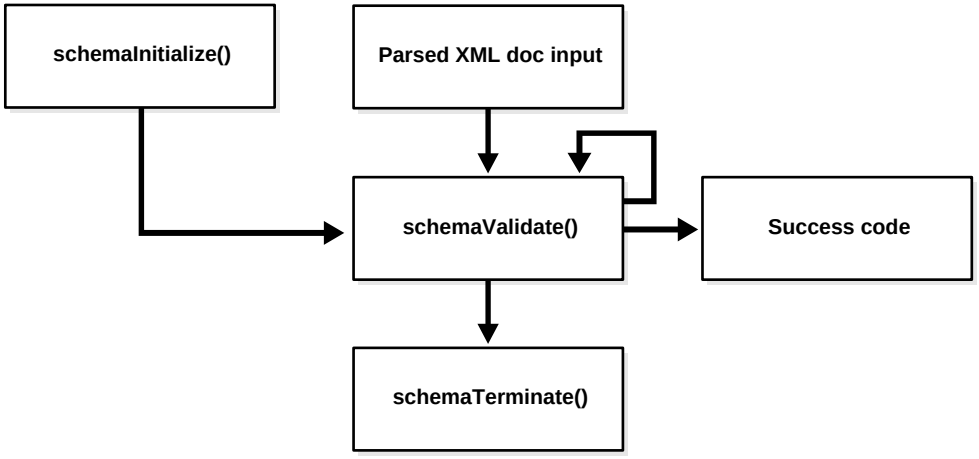
XML Schema Processor for C Usage Diagram

Figure 15–1 describes the calling sequence for the XML Schema Processor for C, as follows:

The sequence of calls to the processor is: initialize, validate, validate,..., validate, terminate.

1. The initialize call is invoked once at the beginning of a session; it returns a Schema context which is used throughout the session.
2. The instance document to be validated is first parsed with the XML parser.
3. The XML context for the instance is then passed to the Schema validate function, along with an optional schema URL.
4. If no explicit schema is defined in the instance document, the default schema will be used.
5. More documents may then be validated using the same schema context.
6. When the session is over, the Schema tear down function is called, which releases all memory allocated by the loaded schemas.

Figure 15–1 XML Schema Processor for C Usage Diagram



How to Run XML Schema for C Sample Programs

This directory contains a sample XML Schema application that illustrates how to use Oracle XML Schema Processor with its API. Table 15–3 lists the provided sample files.

Table 15–3 XML Schema for C Samples Provided

Sample File	Description
Makefile	Makefile to build the sample programs and run them, verifying correct output.
xsdtest.c	Trivial program which invokes the XML Schema for C API
car.{xsd,xml,std}	Sample schema, instance document, and expected output respectively, after running `xsdtest` on them.
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, run `make sure`.

16

XML Parser for C++

This chapter contains the following sections:

Accessing XML Parser for C++

The XML Parser for C++ is provided with Oracle9*i* and Oracle9*i* Application Serverand is also available for download from the OTN site: `http://otn.oracle.com/tech/xml.`

It is located at `$ORACLE_HOME/xdk/cpp/parser`.

XML Parser for C++ Features

`readme.html` in the root directory of the software archive contains release specific information including bug fixes and API additions.

XML Parser for C++ will check if an XML document is well-formed, and optionally validate it against a DTD. The parser will construct an object tree which can be accessed through a DOM interface or operate serially through a SAX interface.

You can post questions, comments, or bug reports to the XML Discussion Forum at `http://otn.oracle.com/tech/xml/ .`

Specifications

See the following:

See Also:

- The `doc` directory in your install area
- *Oracle9i XML API Reference - XDK and Oracle XML DB*
- `http://otn.oracle.com/tech/xml/`

Memory Allocation

The memory callback functions `memcb` may be used if you wish to use your own memory allocation. If they are used, all of the functions should be specified.

The memory allocated for parameters passed to the SAX callbacks or for nodes and data stored with the DOM parse tree will not be freed until one of the following is done:

- `xmlclean()` is called.
- `xmlterm()` is called.

Thread Safety

If threads are forked off somewhere in the midst of the init-parse-term sequence of calls, you will get unpredictable behavior and results.

Data Types Index

Table 16–1 lists the datatypes used in XML Parser for C++.

Table 16–1 Datatypes Used in XML Parser for C++

DataType	Description
oratext	String pointer
xmlctx	Master XML context
xmlmemcb	Memory callback structure (optional)
xmlsaxcb	SAX callback structure (SAX only)
ub4	32-bit (or larger) unsigned integer
uword	Native unsigned integer

Error Message Files

Error message files are provided in the mesg/ subdirectory. The messages files also exist in the `$ORACLE_HOME/xdk/mesg` directory. You may set the environment variable `ORA_XML_MESG` to point to the absolute path of the `mesg/` subdirectory although this not required.

Validation Modes

> **See Also:** Available validation modes are described in "Oracle XML Parsers Validation Modes" on page 4-5.

XML Parser for C++ Usage

Figure 16–1 illustrates the XML Parser for C++ functionality.

1. `xmlinit()` function initializes the parsing process.
2. The XML input can be either an XML file or string buffer. This inputs the following methods:
   - `XMLParser.xmlparse()` if the input is an XML file

- `XMLParser.xmlparseBuffer()` if the input is a string buffer

3. DOM or SAX API

   DOM: If you are using the DOM interface, include the following steps:

   - The `XMLParser.xmlparse()` or `.xmlparserBuffer()` method calls `.getDocument Element()`. If no other DOM methods are being applied, you can invoke `.xmlterm()`.
   - This optionally calls other DOM methods if required. These are typically Node class methods or print methods. It outputs the DOM document.
   - If complete, the process invokes `.xmlterm()`
   - You can first invoke `.xmlclean()` to clean up any data structure created during the parse process. You would then call `.xmlterm()`

   SAX: If you are using the SAX interface, include the following steps:

   - Process the results of the parser from `.xmlparse()` or `.xmlparseBuffer()` through callback methods.
   - Register the callback methods

4. Use `.xmlclean()` to clean up the memory and structures used during a parse, and go to Step 5. or return to Step 2.
5. Terminate the parsing process with `.xmlterm()`

Parser Calling Sequence

The sequence of calls to the parser can be any of the following:

- `XMLParser.xmlinit()` - `XMLParser.xmlparse()` or

  `XMLParser.xmlparsebuf()` - `XMLParser.xmlterm()`

- `XMLParser.xmlinit()` - `XMLParser.xmlparse()` or

  `XMLParser.xmlparsebuf()` - `XMLParser.xmlclean()` - `XMLParser.xmlparse()` or

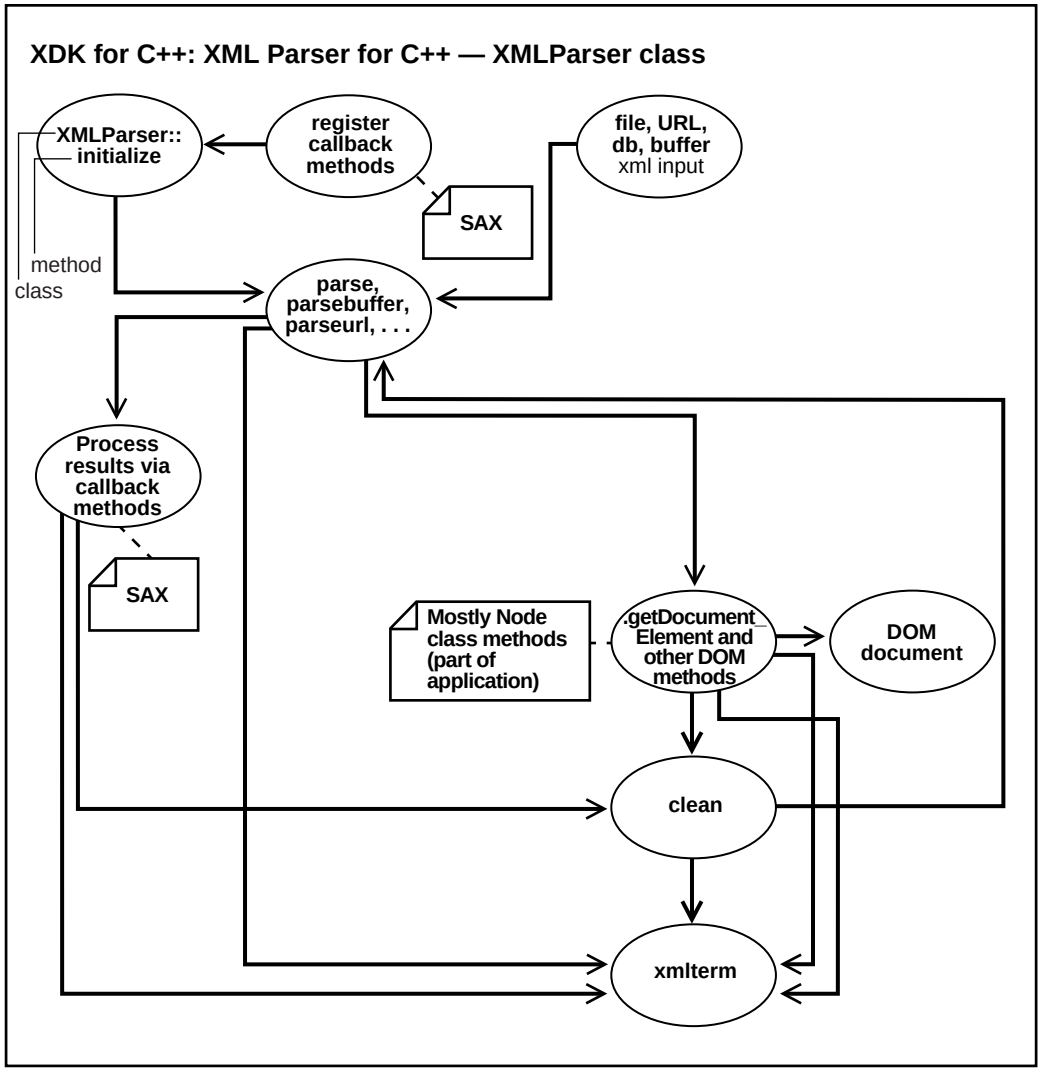
  `XMLParser.xmlparsebuf()` - `XMLParser.xmlclean()` -... - `XMLParser.xmlterm()`

- `XMLParser.xmlinit()` - `XMLParser.xmlparse()` or

  `XMLParser.xmlparsebuf()` - `XMLParser.xmlparse()` or

  `XMLParser.xmlparsebuf()` -... - `XMLParser.xmlterm()`

Figure 16–1 XML Parser for C++ (DOM and SAX Interfaces) Usage



XML Parser for C++ Default Behavior

The following is the XML Parser for C++ default behavior:

- Character set encoding is UTF-8. If all your documents are ASCII, you are encouraged to set the encoding to US-ASCII for better performance.
- Messages are printed to stderr unless msghdlr is given.
- A parse tree which can be accessed by DOM APIs is built unless saxcb is set to use the SAX callback APIs. Note that any of the SAX callback functions can be set to NULL if not needed.
- The default behavior for the parser is to check that the input is well-formed but not to check whether it is valid. The flag XML_FLAG_VALIDATE can be set to validate the input. The default behavior for whitespace processing is to be fully conformant to the XML 1.0 spec, that is, all whitespace is reported back to the application but it is indicated which whitespace is ignorable. However, some applications may prefer to set the XML_FLAG_DISCARD_WHITESPACE which will discard all whitespace between an end-element tag and the following start-element tag.

Note: It is recommended that you set the default encoding explicitly if using only single byte character sets (such as US-ASCII or any of the ISO-8859 character sets) for performance up to 25% faster than with multibyte character sets, such as UTF-8.

DOM and SAX APIs

Oracle XML parser for C++ checks if an XML document is well-formed, and optionally validates it against a DTD. The parser constructs an object tree which can be accessed through one of the following interfaces:

- DOM interface
- Serially through a SAX interface

These two XML APIs:

- DOM: Tree-based APIs. A tree-based API compiles an XML document into an internal tree structure, then allows an application to navigate that tree using the Document Object Model (DOM), a standard tree-based API for XML and HTML documents.
- SAX: Event-based APIs. An event-based API, on the other hand, reports parsing events (such as the start and end of elements) directly to the application through callbacks, and does not usually build an internal tree. The application implements handlers to deal with the different events, much like handling events in a graphical user interface.

Tree-based APIs are useful for a wide range of applications, but they often put a great strain on system resources, especially if the document is large (under very controlled circumstances, it is possible to construct the tree in a lazy fashion to avoid some of this problem). Furthermore, some applications need to build their own, different data trees, and it is very inefficient to build a tree of parse nodes, only to map it onto a new tree.

In both of these cases, an event-based API provides a simpler, lower-level access to an XML document: you can parse documents much larger than your available system memory, and you can construct your own data structures using your callback event handlers.

Using the SAX API

To use SAX, an `xmlsaxcb` structure is initialized with function pointers and passed to the `xmlinit()` call. A pointer to a user-defined context structure can also be included. That context pointer will be passed to each SAX function.

SAX Callback Structure

The SAX callback structure:

```
typedef struct
```

```
{
 sword (*startDocument)(void *ctx);
 sword (*endDocument)(void *ctx);
 sword (*startElement)(void *ctx, const oratext *name,
             const struct xmlarray *attrs);
 sword (*endElement)(void *ctx, const oratext *name);
 sword (*characters)(void *ctx, const oratext *ch, size_t len);
 sword (*ignorableWhitespace)(void *ctx, const oratext *ch, size_t len);
 sword (*processingInstruction)(void *ctx, const oratext *target,
             const oratext *data);
 sword (*notationDecl)(void *ctx, const oratext *name,
             const oratext *publicId, const oratext *systemId);
 sword (*unparsedEntityDecl)(void *ctx, const oratext *name,
             const oratext *publicId,
             const oratext *systemId, const oratext *notationName);
 sword (*nsStartElement)(void *ctx, const oratext *qname,
             const oratext *local, const oratext *nsp,
             const struct xmlnodes *attrs);
} xmlsaxcb;
```

Invoking XML Parser for C++

XML Parser for C++ can be invoked in two ways:

- By invoking the executable on the command line
- By writing C++ code and using the supplied APIs

Command Line Usage

The XML Parser for C++ can be called as an executable by invoking `bin/xml`

Table 16–2 lists the command line options.

Table 16–2 XML Parser for C++: Command Line Options

Option	Description
-c	Conformance check only, no validation
-e encoding	Specify input file encoding
-h	Help - show this usage help
-n	Number - DOM traverse and report number of elements
-p	Print document and DTD structures after parse

Table 16–2 XML Parser for C++: Command Line Options

Option	Description
-x	Exercise SAX interface and print document
-v	Version - display parser version then exit
-w	Whitespace - preserve all whitespace

Writing C++ Code to Use Supplied APIs

XML Parser for C++ can also be invoked by writing code to use the supplied APIs. The code must be compiled using the headers in the include/ subdirectory and linked against the libraries in the lib/ subdirectory. Please see the `Makefile` in the sample/ subdirectory for full details of how to build your program.

Using the Sample Files Included with Your Software

$ORACLE_HOME/xdk/cpp/parser/`sample/` directory contains several XML applications to illustrate how to use the XML Parser for C++ with the DOM and SAX interfaces.

Table 16–3 lists the sample files in sample/ directory.

Table 16–3 XML Parser for C++ sample/ Files

sample/ File Name	Description
DOMNamespace.cpp	Source for DOMNamespace program
DOMNamespace.std	Expected output from DOMNamespace
DOMSample.cpp	Source for DOMSample program
DOMSample.std	Expected output from DOMSample
FullDOM.c	Sample usage of DOM interface
FullDOM.std	Expected output from FullDOM
Make.bat	Batch file to build sample executables
Makefile	Makefile for sample programs
NSExample.xml	Sample XML file using namespaces
SAXNamespace.cpp	Source for SAXNamespace program
SAXNamespace.std	Expected output from SAXNamespace

Table 16–3 XML Parser for C++ sample/ Files (Cont.)

sample/ File Name	Description
SAXSample.cpp	Source for SAXSample program
SAXSample.std	Expected output from SAXSample
XSLSample.cpp	Source for XSLSample program
XSLSample.std	Expected output from XSLSample
class.xml	XML file that may be used with XSLSample
iden.xsl	Stylesheet that may be used with XSLSample
cleo.xml	XML version of Shakespeare's play

Running the XML Parser for C++ Sample Programs

Building the Sample Programs

Change directories to the sample directory (`$ORACLE_HOME/xdk/demo/cpp/parser` on Solaris™ Operating Environment) and read the README file. This will explain how to build the sample programs according to your platform.

Sample Programs

Table 16–4 lists the programs built by the sample files in sample/.

Table 16–4 XML Parser for C++, Sample Programs Built in sample/

Built Program	Description
SAXSample	A sample application using SAX APIs. Prints out all speakers in each scene, that is, all the unique SPEAKER elements within each SCENE element.
DOMSample [speaker]	A sample application using DOM APIs. Prints all speeches made by the given speaker. If no speaker is specified, "Soothsayer" is used. Note that major characters have uppercase names (for example, "CLEOPATRA"), whereas minor characters have capitalized names (for example, "Attendant"). See the output of SAXSample.
SAXNamespace	A sample application using Namespace extensions to SAX API; prints out all elements and attributes of NSExample.xml along with full namespace information.

Table 16–4 XML Parser for C++, Sample Programs Built in sample/ (Cont.)

Built Program	Description
DOMNamespace	Same as SAXNamespace except using DOM interface.
FullDOM	Sample usage of full DOM interface. Exercises all the calls, but does nothing too exciting.
XSLSample <xmlfile> <xsl ss>	Sample usage of XSL processor. It takes two filenames as input, the XML file and the XSL stylesheet. Note: If you redirect stdout of this program to a file, you may encounter some missing output, depending on your environment.

17

XSLT Processor for C++

This chapter contains the following sections:

Accessing XSLT for C++

XSLT for C++ is provided with Oracle9*i* and Oracle9*i* Application Server. It is also available for download from the OTN site: `http://otn.oracle.com/tech/xml`.

It is located at `$ORACLE_HOME/xdk/cpp/parser`.

XSLT for C++ Features

`readme.html` in the root directory of the software archive contains release specific information including bug fixes and API additions.

You can post questions, comments, or bug reports to the XML Discussion Forum at `http://otn.oracle.com/tech/xml`.

Specifications

See the following:

See Also:

- The doc directory in your install area
- *Oracle9i XML API Reference - XDK and Oracle XML DB*

XSLT for C++ (DOM Interface) Usage

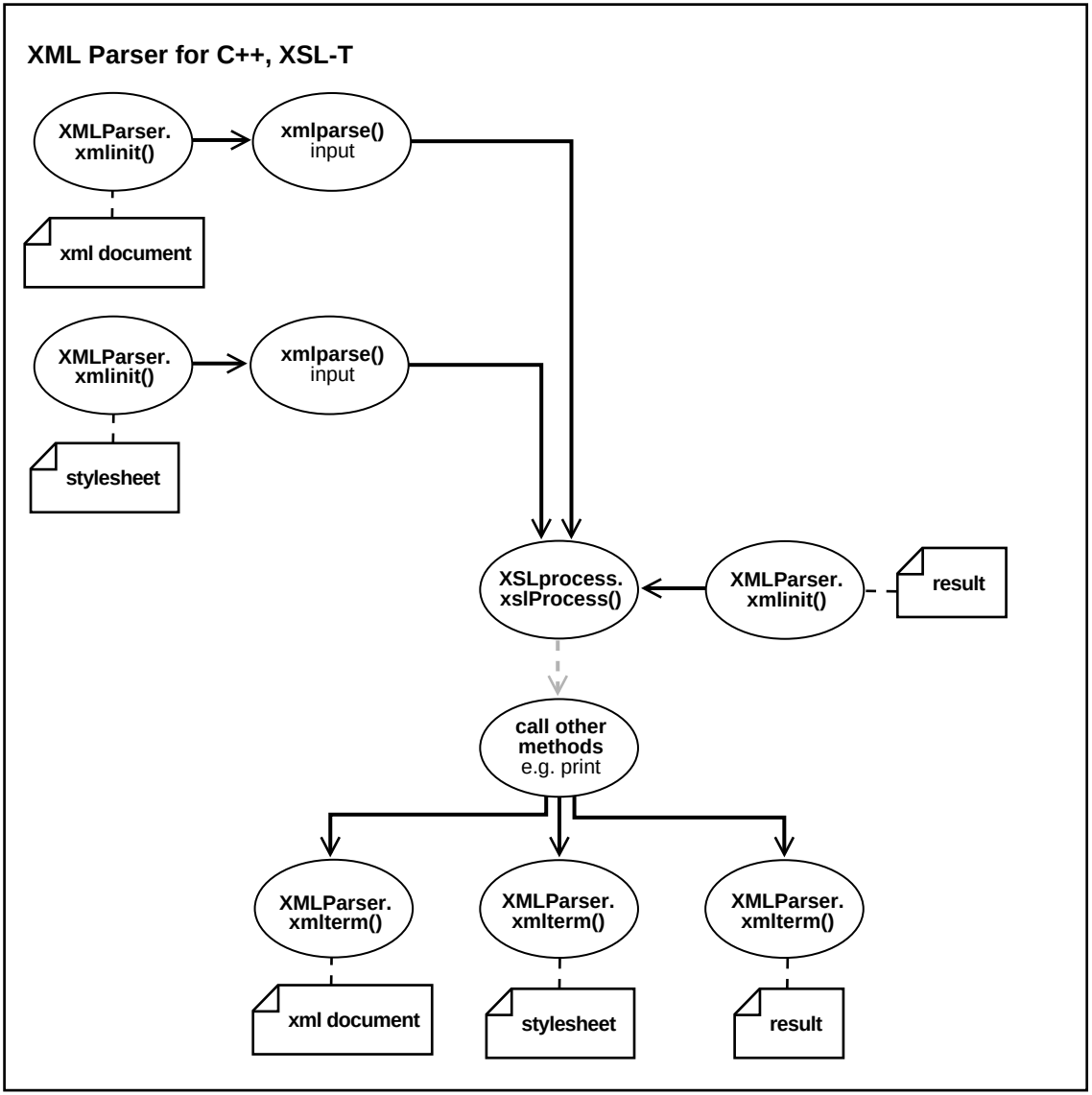
Figure 17–1 shows the XSLT for C++ functionality for the DOM interface.

1. There are two inputs to XMLParser.`xmlparse()`:
   - The stylesheet to be applied to the XML document
   - XML document

   The output of XMLParser.`xmlparse()`, the parsed stylesheet and parsed XML document are sent to the XSLProcess.`xslprocess()` method for processing.
2. `XMLParser.xmlinit()` initializes the XSLT processing. XMLParser. `xmlinit() also` initializes the `xslprocess()` result.
3. `XSLProcess.xslProcess()` optionally calls other methods, such as print methods. You can see the list of available methods either on OTN or in *Oracle9i XML API Reference - XDK and Oracle XML DB*.

4. The resultant document (XML, HTML, VML, and so on) is typically sent to an application for further processing.
5. The application terminates the XSLT process by declaring `XMLParser.xmlterm()` for the XML document, stylesheet, and final result.

Figure 17–1 XSLT for C++ Functionality (DOM Interface) Usage



Invoking XSLT for C++

XSLT for C++ can be invoked in two ways:

- By invoking the executable on the command line
- By writing C++ code and using the supplied APIs

Command Line Usage

The XSLT for C++ can be called as an executable by invoking `bin/xml`

Table 17–1 lists the command line options.

Table 17–1 XXSLT for C++: Command Line Options

Option	Description
-e encoding	Specify input file encoding
-h	Help - show this usage help
-v	Version - display parser version then exit
-w	Whitespace - preserve all whitespace
-s	Stylesheet

Writing C++ Code to Use Supplied APIs

XXSLT for C++ can also be invoked by writing code to use the supplied APIs. The code must be compiled using the headers in the include/ subdirectory and linked against the libraries in the lib/ subdirectory. Please see the `Makefile` in the sample/ subdirectory for full details of how to build your program.

Using the Sample Files Included with Your Software

$ORACLE_HOME/xdk/cpp/parser/`sample/` directory contains several XML applications to illustrate how to use the XXSLT for C++.

Table 17–2 lists the sample files in sample/ directory.

Table 17–2 XML Parser for C++ sample/ Files

sample/ File Name	Description
XSLSample.cpp	Source for XSLSample program

Table 17–2 XML Parser for C++ sample/ Files(Cont.)

sample/ File Name	Description
XSLSample.std	Expected output from XSLSample
class.xml	XML file that may be used with XSLSample
iden.xsl	Stylesheet that may be used with XSLSample
cleo.xml	XML version of Shakespeare's play

Running the XSLT for C++ Sample Programs

Building the Sample programs

Change directories to the sample directory and read the README file. This will explain how to build the sample programs according to your platform.

Sample Programs

Table 17–3 lists the programs built by the sample files.

Table 17–3 XML Parser for C++, Sample Programs Built in sample/

Built Program	Description
XSLSample <xmlfile> <xsl ss>	Sample usage of XSL processor. It takes two filenames as input, the XML file and the XSL stylesheet. Note: If you redirect stdout of this program to a file, you may encounter some missing output, depending on your environment.

18

XML Schema Processor for C++

This chapter contains the following sections:

Oracle XML Schema Processor for C++ Features

The XML Schema Processor for C++ is a companion component to the XML Parser for C++ that allows support to simple and complex datatypes into XML applications with Oracle9i.

The XML Schema Processor for C++ supports the W3C XML Schema Recommendation, with the goal being that it be 100% fully conformant when XML Schema becomes a W3C Recommendation. This makes writing custom applications that process XML documents straightforward in the Oracle9i environment, and means that a standards-compliant XML Schema Processor is part of the Oracle9i platform on every operating system where Oracle9i is ported.

> **See Also:** Chapter 4, "XML Parser for Java", for more information about XML Schema and why you would want to use XML Schema.

Oracle XML Schema for C++ Features

XML Schema Processor for C++ has the following features:

- Supports simple and complex types
- Built upon the XML Parser for C++
- Supports the W3C XML Schema Recommendation

The XML Schema Processor for C++ class is `XMLSchema`.

> **See Also:** *Oracle9i XML API Reference - XDK and Oracle XML DB*

Online Documentation

Documentation for Oracle XML Schema Processor for C++ is located in the `doc` directory in your install area.

Standards Conformance

The Schema Processor conforms to the following standards:

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

XML Schema Processor for C++: Provided Software

Table 18–1 lists the supplied files and directories with this release:

Table 18–1 XML Schema Processor for C++: Supplied Files

Directory an d Files	Description
license.html	Licensing agreement
readme.html	This file
bin	Schema processor executable, "schema"
doc	API documentation
include	header files
lib	XML / XSL / Schema & support libraries
mesg	Error message files
sample	Example usage of the Schema processor

Table 18–2 lists the included libraries:

Table 18–2 XML Schema Processor for C++: Supplied Libraries

Included Library	Description
libxml9.a	XML Parser / XSL Processor
libxsd9.a	XML Schema Processor
libcore9.a	CORE functions
libnls9.a	Globalization Support

Invoking XML Schema Processor for C++

The XML Schema Processor can be called as an executable by invoking `bin/schema` in the install area. This takes two arguments:

- XML instance document
- Optionally, a default schema

The Schema processor can also be invoked by writing code using the supplied APIs. The code must be compiled using the headers in the include / subdirectory and linked against the libraries in the lib / subdirectory. See Makefile in the sample / subdirectory for details on how to build your program.

An error message file is provided in the mesg/ subdirectory. Currently, the only message file is in English although message files for other languages may be supplied in future releases.

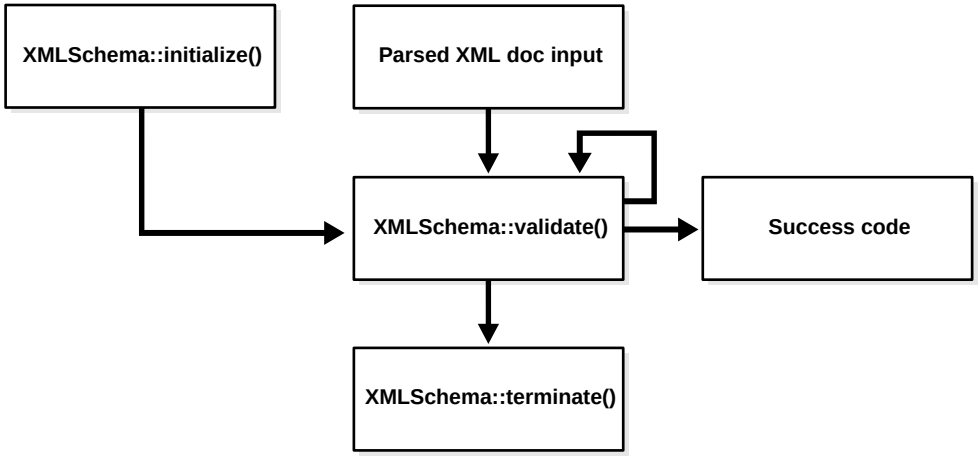
XML Schema Processor for C++ Usage Diagram

Figure 18–1 illustrates the calling sequence of XMl Schema Processor for C++, as follows:

The sequence of calls to the processor is: initialize, validate, validate,..., validate, terminate.

1. The initialize call is invoked once at the beginning of a session; it returns a Schema context which is used throughout the session.
2. The instance document to be validated is first parsed with the XML parser.
3. The XML context for the instance is then passed to the Schema validate function, along with an optional schema URL.
4. If no explicit schema is defined in the instance document, the default schema will be used.
5. More documents may then be validated using the same schema context.
6. When the session is over, the Schema tear down function is called, which releases all memory allocated by the loaded schemas.

Figure 18–1 XML Schema Processor for C++ Usage Diagram



Running the Provided XML Schema Sample Programs

This directory contains a sample XML Schema application that illustrates how to use Oracle XML Schema Processor with its API. Table 18–3 lists the provided sample files.

Table 18–3 XML Schema 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's, instance document, expected output respectively, after running xsdtest on them.
pub.{xsd,xml,std}	Third sample Schema's, 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, run `make sure`.

19

XML Class Generator for C++

This chapter contains the following sections:

Accessing XML C++ Class Generator

The XML C++ Class Generator is provided with Oracle9*i* and is also available for download from the OTN site:

```
http://otn.oracle.com/tech/xml
```

It is located in `$ORACLE_HOME/xdk/cpp/classgen`. Information about using the Class Generator is available with the software.

Using XML C++ Class Generator

The XML C++ Class Generator creates source files from an XML DTD or XML Schema. The Class Generator takes the Document Type Definition (DTD) or the XML Schema, and generates classes for each defined element. Those classes are then used in a C++ program to construct XML documents conforming to the DTD.

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 in conjunction with the Oracle XML Parser for C++, which parses the input and passes the parsed document to the class generator.

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. See "xmlcg Usage" on page 19-5.

Error Message Files

Error message files are provided in the mesg/ subdirectory. The messages files also exist in the `$ORACLE_HOME/xdk/mesg` directory. You may set the environment variable `ORA_XML_MESG` to point to the absolute path of the `mesg/` subdirectory although this not required.

XML C++ Class Generator Usage

Figure 19–1 summarizes the XML C++ Class Generator usage.

1. From the bin directory, at the command line, enter the following:

   ```
   xml [XML document file name, such as xxxxx]
   ```

   where XML document file name is the name of the parsed XML document or parsed DTD being processed. The XML document must have an associated DTD.

   The Input to the XML C++ Class Generator is an XML document containing a DTD, or an external DTD. The document body itself is ignored; only the DTD is relevant, though the document must conform to the DTD.

   Accepted character set encoding for input files are listed in "Input to the XML C++ Class Generator" on page 19-3.

2. Two source files are output, a xxxxx.h header file and a xxxxx.cpp C++ file. These are named after the DTD file.

3. The output files are typically used to generate XML documents.

Constructors are provided for each class (element) that allow an object to be created in the following two ways:

- Initially empty, then adding the children or data after the initial creation
- Created with the initial full set of children or initial data

A method is provided for #PCDATA (and Mixed) elements to set the data and, when appropriate, set an element's attributes.

Input to the XML C++ Class Generator

Input is an XML document containing a DTD. The document body itself is ignored; only the DTD is relevant, though the dummy document must conform to the DTD. The underlying XML parser only accepts file names for the document and associated external entities. In future releases, no dummy document will be required, and URIs for additional protocols will be accepted.

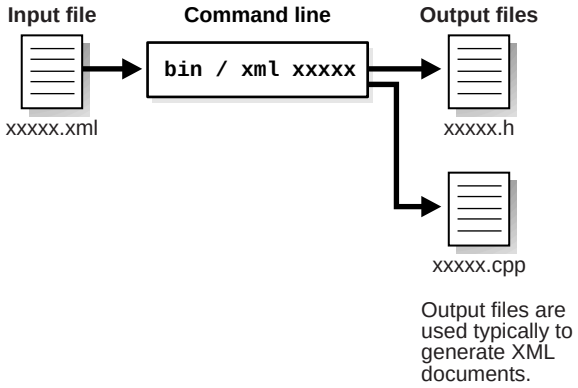
Character Set Support

The following lists supported Character Set Encoding for files input to XML C++ Class Generator. These are in addition to the character sets specified in Appendix A, "Character Sets", of *Oracle9i Database Globalization Support Guide*.

- BIG 5
- EBCDIC-CP-*
- EUC-JP
- EUC-KR
- GB2312
- ISO-2022-JP
- ISO-2022-KR
- ISO-8859-1, ISO-8859-2, ISO-8859-3, ..., ISO-8859-9
- ISO-10646-UCS-2
- ISO-10646-UCS-4
- KOI8-R
- Shift_JIS
- US-ASCII
- UTF-8
- UTF-16

Default: The default encoding is UTF-8. It is recommended that you set the default encoding explicitly if using only single byte character sets (such as US-ASCII or any of the ISO-8859 character sets) for performance up to 25% faster than with multibyte character sets, such as UTF-8.

Figure 19–1 XML C++ Class Generator Functionality



xmlcg Usage

The standalone parser may be called as an executable by invoking bin/xmlcg. For example:

```
xmlcg [flags] <XML document or External DTD>
```

Table 19–1 lists the xmlcg optional flags.

Table 19–1 xmlcg Optional Flags

xmlcg Optional Flags	Description
-d name	DTD - Input is an external DTD with the given name
-o directory	Output directory for generated files (default is current directory)
-e encoding	Encoding - Default input file encoding
-h	Help - Show this usage help
-v	Version - Show the Class Generator version

Using the XML C++ Class Generator Examples in sample

Table 19–2 lists the files supplied the sample XML C++ Class Generator `sample` directory.

Table 19–2 XML C++ Class Generator Examples in sample/

Sample File Name	Description
CG.cpp	Sample program
CG.xml	XML file contains DTD and dummy document
CG.dtd	DTD file referenced by CG.xml
Make.bat on Windows NT Makefile on UNIX	Batch file (on Windows NT) or script file (on UNIX) to generate classes and build the sample programs.
README	A readme file with these instructions

The `make.bat` batch file (on Windows NT) or `Makefile` (on UNIX) do the following:

- Generate classes based on CG.xml into Sample.h and Sample.cpp
- Compile the program CG.cpp (using Sample.h), and link this with the Sample object into an executable named CG.exe in the...\bin (or .../bin) directory.

XML C++ Class Generator Example 1: XML — Input File to Class Generator, CG.xml

This XML file, CG.xml, inputs XML C++ Class Generator. It references the DTD file, CG.dtd.

```
<?xml version="1.0"?>
<!DOCTYPE Sample SYSTEM "CG.dtd">
  <Sample>
    <B>Be!</B>
    <D attr="value"></D>
    <E>
      <F>Formula1</F>
      <F>Formula2</F>
    </E>
  </Sample>
```

XML C++ Class Generator Example 2: DTD — Input File to Class Generator, CG.dtd

This DTD file, CG.dtd is referenced by the XML file CG.xml. CG.xml inputs XML C++ Class Generator.

```
<!ELEMENT Sample (A | (B, (C | (D, E))) | F)>
<!ELEMENT A (#PCDATA)>
<!ELEMENT B (#PCDATA | F)*>
```

```
<!ELEMENT C (#PCDATA)>
<!ELEMENT D (#PCDATA)>
<!ATTLIST D attr CDATA #REQUIRED>
<!ELEMENT E (F, F)>
<!ELEMENT F (#PCDATA)>
```

XML C++ Class Generator Example 3: CG Sample Program

The CG sample program, `CG.cpp`, does the following:

1. Initializes the XML parser
2. Loads the DTD (by parsing the DTD-containing file-- the dummy document part is ignored)
3. Creates some objects using the generated classes
4. Invokes the validation function which verifies that the constructed classes match the DTD
5. Writes the constructed document to Sample.xml

```
////////////////////////////////////////////////////////////////////////
// NAME        CG.cpp
// DESCRIPTION Demonstration program for C++ Class Generator usage
////////////////////////////////////////////////////////////////////////

#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
    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
```

Part IV

XDK for PL/SQL

These chapters describe how to access and use Oracle XML Developer's Kit (XDK) for PL/SQL:

- Chapter 20, "XML Parser for PL/SQL"
- Chapter 21, "XSLT Processor for PL/SQL"
- Chapter 22, "XML Schema Processor for PL/SQL"
- Chapter 23, "XSU for PL/SQL"

Note: In Oracle9*i*, XML-SQL Utility (XSU) for PL/SQL is considered part of the XDK for PL/SQL. In this manual, XSU is described in Chapter 8, "XML SQL Utility (XSU)".

20

XML Parser for PL/SQL

This chapter contains the following sections:

Accessing XML Parser for PL/SQL

XML Parser for PL/SQL is provided with Oracle9*i* and is also available for download from the OTN site: `http://otn.oracle.com/tech/xml`.

It is located at `$ORACLE_HOME/xdk/plsql/parser`

What's Needed to Run XML Parser for PL/SQL

Appendix B, "XDK for PL/SQL: Specifications" lists the specifications and requirements for running the XML Parser for PL/SQL. It also includes syntax cheat sheets.

Using XML Parser for PL/SQL (DOM Interface)

The XML Parser for PL/SQL makes developing XML applications with Oracle9*i* a simplified and standardized process. With the PL/SQL interface, Oracle shops familiar with PL/SQL can extend existing applications to take advantage of XML as needed.

Since the XML Parser for PL/SQL is implemented in PL/SQL and Java, it can run "out of the box" on the Oracle9*i* Java Virtual Machine.

XML Parser for PL/SQL supports the W3C XML 1.0 specification. The goal is to be 100% conformant. It can be used both as a validating or non-validating parser.

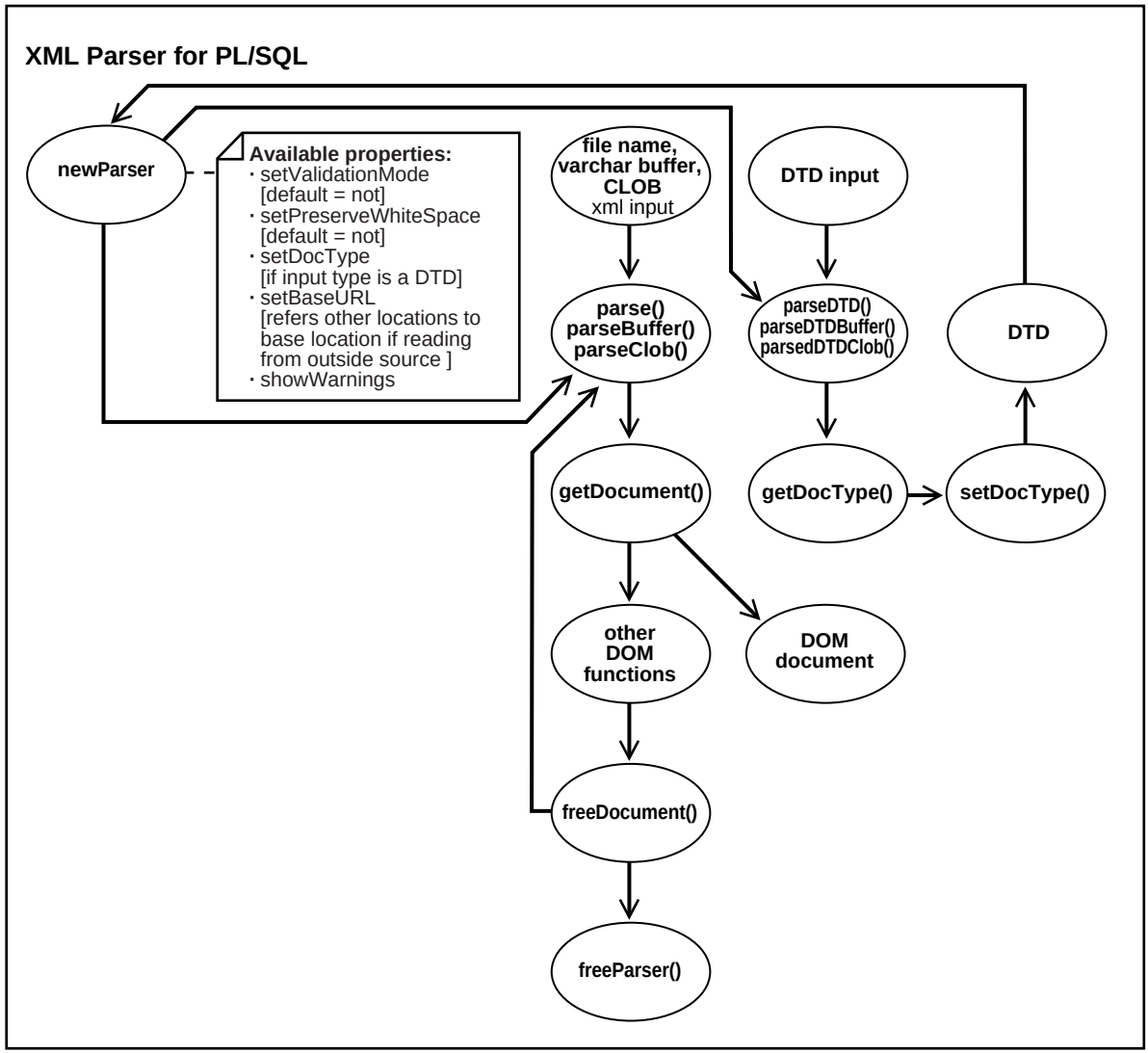
In addition, XML Parser for PL/SQL provides the two most common APIs you need for processing XML documents:

- W3C-recommended Document Object Model (DOM)
- XSLT and XPath recommendations

This makes writing custom applications that process XML documents straightforward in the Oracle9*i* environment, and means that a standards-compliant XML parser is part of the Oracle9*i* platform on every operating system where Oracle9*i* is ported.

Figure 20–1 shows the XML Parser for PL/SQL usage and parsing process diagram.

Figure 20–1 XML Parser for PL/SQL Functionality (DOM Interface)



1. Make a newParser declaration to begin the parsing process for the XML document and DTD, if applicable.

   Table 20–1 lists available properties for the `newParser` procedure:

Table 20–1 XML Parser for PL/SQL: newParser() Properties

Property	Description
setValidationMode	Default = Not
setPreserveWhiteSpace	Default = Not
setDocType	Use if input type is a DTD
setBaseURL	Refers to other locations to the base locations, if reading from an outside source
showWarnings	Turns warnings on or off.

2. The XML and DTD can be input as a file, varchar buffer, or CLOB. The XML input is called by the following procedures:
   - `parse()` if the XML input is a file
   - `parseBuffer()` if the XML input is an varchar buffer
   - `parserClob()` if the XML input is a CLOB

   If a DTD is also input, it is called by the following procedures:
   - `parseDTD()` if the input is an DTD file
   - `parseDTDBuffer()` if the DTD input is an varchar buffer
   - `parserDTDClob()` if the DTD input is a CLOB

For the XML Input: For an XML input, the parsed result from `Parse()`, `ParserBuffer()`, or `ParserClob()` procedures is sent to `GetDocument()`.

3. `getDocument()` procedure performs the following:
   - Outputs the parsed XML document as a DOM document typically to be used in a PL/SQL application, or
   - Applies other DOM functions, if applicable.
4. Use `freeDocument()` function to free up the parser and parse the next XML input
5. Use `freeParser()` to free up any temporary document structures created during the parsing process

For the DTD input: The parsed result from `parseDTD()`, `parseDTDBuffer()`, or `parseDTDClob()` is used by `getDocType()` function.

6. `getDocType()` then uses `setDocType()` to generate a DTD object.

7. The DTD object can be fed into the parser using `setDocType()` to override the associated DTD.

See Also:

- *Oracle9i XML API Reference - XDK and Oracle XML DB* for a list of available optional DOM functions.
- *Oracle9i XML Database Developer's Guide - Oracle XML DB* , the chapter on the PL/SQL API for XMLType.

XML Parser for PL/SQL: Default Behavior

The following is the default behavior for XML Parser for PLSQL XML:

- A parse tree which can be accessed by DOM APIs is built
- The parser is validating if a DTD is found, otherwise it is non-validating
- Errors are not recorded unless an error log is specified; however, an application error will be raised if parsing fails

The types and methods described in this manual are supplied with the PLSQL package xmlparser().

Using XML Parser for PL/SQL Examples in the Sample Directory

Setting Up the Environment to Run the Sample Programs

The $ORACLE_HOME/xdk/plsql/parser/sample/ directory contains two sample XML applications:

- domsample
- xslsample

These show you how to use XML Parser for PL/SQL.

To run these sample programs carry out the following steps:

1. Load the PL/SQL parser into the database. To do this, follow the instructions given in the README file under the lib directory.
2. You must have the appropriate Java security privileges to read and write from a file on the file system. To this, first startup SQL*Plus (located typically under

$ORACLE_HOME/bin) and connect as a user with administration privileges, such as, 'internal':

For example:

```
% sqlplus
SQL> connect / as sysdba
```

3. A password might be required or the appropriate user with administration privileges. Contact your System Administrator, DBA, or Oracle support, if you cannot login with administration privileges.
4. Give special privileges to the user running this sample. It must be the same one under which you loaded the jar files and plsql files in Step 1.

   For example, for user 'scott':

   ```
   SQL> grant javauserpriv to scott;
   SQL> grant javasyspriv to scott;
   ```

   You should see two messages that say "Grant succeeded." Contact your System Administrator, DBA, or Oracle support, if this does not occur.

   Now, connect again as the user under which the PL/SQL parser was loaded in step 1. For example, for user 'scott' with password 'tiger':

   ```
   SQL> connect scott/tiger
   ```

Running domsample

To run domsample carry out the following steps:

1. Load domsample.sql script under SQL*Plus (if SQL*Plus is not up, first start it up, connecting as the user running this sample) as follows:

   ```
   SQL> @domsample
   ```

   The `domsample.sql` script defines a procedure domsample with the following syntax:

   ```
   domsample(dir varchar2, inpfile varchar2, errfile varchar2)
   ```

   where:

Argument	Description
'dir'	Must point to a valid directory on the external file system and should be specified as a complete path name
'inpfile'	Must point to the file located under 'dir', containing the XML document to be parsed
'errfile'	Must point to a file you wish to use to record errors; this file will be created under 'dir'

2. Execute the domsample procedure inside SQL*Plus by supplying appropriate arguments for 'dir', 'inpfile', and 'errfile'. For example:

   On Unix, you can could do the following:

   ```
   SQL>execute domsample('/private/scott', 'family.xml', 'errors.txt');
   ```

   On Windows NT, you can do the following:

   ```
   SQL>execute domsample('c:\xml\sample', 'family.xml', 'errors.txt');
   ```

   where `family.xml` is provided as a test case

3. You should see the following output:

   - The elements are: family member member member member
   - The attributes of each element are:

     ```
     family:
     lastname = Smith
       member:
       memberid = m1
       member:
       memberid = m2
       member:
       memberid = m3 mom = m1 dad = m2
       member:
       memberid = m4 mom = m1 dad = m2
     ```

Running xslsample

To run xslsample, carry out these steps:

1. Load the `xslsample.sql` script under SQL*Plus (if SQL*Plus is not up, first start it up, connecting as the user running this sample):

   ```
   SQL>@xslsample
   ```

   `xslsample.sql` script defines a procedure `xslsample` with the following syntax:

   ```
   xslsample ( dir varchar2, xmlfile varchar2, xslfile varchar2, resfile
   varchar2, errfile varchar2 )
   ```

   where:

Argument	Description
'dir'	Must point to a valid directory on the external file system and should be specified as a complete path name.
'xmlfile'	Must point to the file located under 'dir', containing the XML document to be parsed.
'xskfile'	Must point to the file located under 'dir', containing the XSL stylesheet to be applied.
'resfile'	Must point to the file located under 'dir' where the transformed document is to be placed.
'errfile'	Must point to a file you wish to use to record errors; this file will be created under 'dir'

2. Execute the `xslsample` procedure inside SQL*Plus by supplying appropriate arguments for 'dir', 'xmlfile', 'xslfile', and 'errfile'.

   For example:

   - On Unix, you can do the following:

     ```
     SQL>execute xslsample('/private/scott', 'family.xml', 'iden.xsl',
     'family.out', 'errors.txt');
     ```

   - On NT, you can do the following:

     ```
     SQL>execute xslsample('c:\xml\sample', 'family.xml', 'iden.xsl',
     'family.out', 'errors.txt');
     ```

3. The provided test cases are: `family.xml` and `iden.xsl`

4. You should see the following output:

```
Parsing XML document c:\/family.xml
Parsing XSL document c:\/iden.xsl
XSL Root element information
Qualified Name: xsl:stylesheet
Local Name: stylesheet
Namespace: http://www.w3.org/XSL/Transform/1.0
Expanded Name: http://www.w3.org/XSL/Transform/1.0:stylesheet
A total of 1 XSL instructions were found in the stylesheet
Processing XSL stylesheet
Writing transformed document
```

5. `family.out` should contain the following:

```
<family lastname="Smith">
<member memberid="m1">Sarah</member>
<member memberid="m2">Bob</member>
<member memberid="m3" mom="m1" dad="m2">Joanne</member>
<member memberid="m4" mom="m1" dad="m2">Jim</member>
</family>
```

You might see a delay in getting the output when executing the procedure for the first time. This is because Oracle JVM performs various initialization tasks before it can execute a Java Stored Procedure (JSP). Subsequent invocations should run quickly.

If you get errors, ensure the directory name is specified as a complete path on the file system

---

Note: SQL directory aliases and shared directory syntax '\ \' are not supported at this time.

Otherwise, report the problem on the XML discussion forum at `http://otn.oracle.com`

XML Parser for PL/SQL Example: XML — family.xml

This XML file inputs `domsample.sql`.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE family SYSTEM "family.dtd">
<family lastname="Smith">
<member memberid="m1">Sarah</member>
```

```
<member memberid="m2">Bob</member>
<member memberid="m3" mom="m1" dad="m2">Joanne</member>
<member memberid="m4" mom="m1" dad="m2">Jim</member>
</family>
```

XML Parser for PL/SQL Example: DTD — family.dtd

This DTD file is referenced by XML file, `family.xml`.

```
<!ELEMENT family (member*)>
<!ATTLIST family lastname CDATA #REQUIRED>
<!ELEMENT member (#PCDATA)>
<!ATTLIST member memberid ID #REQUIRED>
<!ATTLIST member dad IDREF #IMPLIED>
<!ATTLIST member mom IDREF #IMPLIED>
```

XML Parser for PL/SQL Example: PL/SQL — domsample.sql

```
-- This file demonstrates a simple use of the parser and DOM API.
-- The XML file that is given to the application is parsed and the
-- elements and attributes in the document are printed.
-- It shows you how to set the parser options.

set serveroutput on;
create or replace procedure domsample(dir varchar2, inpfile varchar2,
                                      errfile varchar2) is
p xmlparser.parser;
doc xmldom.DOMDocument;

-- prints elements in a document
procedure printElements(doc xmldom.DOMDocument) is
nl xmldom.DOMNodeList;
len number;
n xmldom.DOMNode;

begin
   -- get all elements
   nl := xmldom.getElementsByTagName(doc, '*');
   len := xmldom.getLength(nl);

   -- loop through elements
   for i in 0..len-1 loop
      n := xmldom.item(nl, i);
```

```
      dbms_output.put(xmldom.getNodeName(n) || ' ');
   end loop;

   dbms_output.put_line('');
end printElements;

-- prints the attributes of each element in a document
procedure printElementAttributes(doc xmldom.DOMDocument) is
nl xmldom.DOMNodeList;
len1 number;
len2 number;
n xmldom.DOMNode;
e xmldom.DOMElement;
nnm xmldom.DOMNamedNodeMap;
attrname varchar2(100);
attrval varchar2(100);

begin

   -- get all elements
   nl := xmldom.getElementsByTagName(doc, '*');
   len1 := xmldom.getLength(nl);

   -- loop through elements
   for j in 0..len1-1 loop
      n := xmldom.item(nl, j);
      e := xmldom.makeElement(n);
      dbms_output.put_line(xmldom.getTagName(e) || ':');

      -- get all attributes of element
      nnm := xmldom.getAttributes(n);

     if (xmldom.isNull(nnm) = FALSE) then
        len2 := xmldom.getLength(nnm);

        -- loop through attributes
        for i in 0..len2-1 loop
           n := xmldom.item(nnm, i);
           attrname := xmldom.getNodeName(n);
           attrval := xmldom.getNodeValue(n);
           dbms_output.put(' ' || attrname || ' = ' || attrval);
        end loop;
        dbms_output.put_line('');
     end if;
   end loop;
```

```
end printElementAttributes;

begin

-- new parser
   p := xmlparser.newParser;

-- set some characteristics
   xmlparser.setValidationMode(p, FALSE);
   xmlparser.setErrorLog(p, dir || '/' || errfile);
   xmlparser.setBaseDir(p, dir);

-- parse input file
   xmlparser.parse(p, dir || '/' || inpfile);

-- get document
   doc := xmlparser.getDocument(p);

-- Print document elements
   dbms_output.put('The elements are: ');
   printElements(doc);

-- Print document element attributes
   dbms_output.put_line('The attributes of each element are: ');
   printElementAttributes(doc);

-- deal with exceptions
exception

when xmldom.INDEX_SIZE_ERR then
   raise_application_error(-20120, 'Index Size error');

when xmldom.DOMSTRING_SIZE_ERR then
   raise_application_error(-20120, 'String Size error');

when xmldom.HIERARCHY_REQUEST_ERR then
   raise_application_error(-20120, 'Hierarchy request error');

when xmldom.WRONG_DOCUMENT_ERR then
   raise_application_error(-20120, 'Wrong doc error');

when xmldom.INVALID_CHARACTER_ERR then
   raise_application_error(-20120, 'Invalid Char error');
```

```
when xmldom.NO_DATA_ALLOWED_ERR then
   raise_application_error(-20120, 'Nod data allowed error');

when xmldom.NO_MODIFICATION_ALLOWED_ERR then
   raise_application_error(-20120, 'No mod allowed error');

when xmldom.NOT_FOUND_ERR then
   raise_application_error(-20120, 'Not found error');

when xmldom.NOT_SUPPORTED_ERR then
   raise_application_error(-20120, 'Not supported error');

when xmldom.INUSE_ATTRIBUTE_ERR then
   raise_application_error(-20120, 'In use attr error');

end domsample;
/
show errors;
```

XML Parser for PL/SQL Example: PL/SQL — xslsample.sql

```
-- This file demonstates a simple use of XSLT transformation capabilities.
-- The XML and XSL files that are given to the application are parsed,
-- the transformation specified is applied and the transformed document is
-- written to a specified result file.
-- It shows you how to set the parser options.

set serveroutput on;
create or replace procedure xslsample(dir varchar2, xmlfile varchar2,
                                      xslfile varchar2, resfile varchar2,
                                      errfile varchar2) is
p xmlparser.Parser;
xmldoc xmldom.DOMDocument;
xmldocnode xmldom.DOMNode;
proc xslprocessor.Processor;
ss xslprocessor.Stylesheet;
xsldoc xmldom.DOMDocument;
docfrag xmldom.DOMDocumentFragment;
docfragnode xmldom.DOMNode;
xslelem xmldom.DOMElement;
nspace varchar2(50);
xslcmds xmldom.DOMNodeList;

begin
```

```
-- new parser
   p := xmlparser.newParser;

-- set some characteristics
   xmlparser.setValidationMode(p, FALSE);
   xmlparser.setErrorLog(p, dir || '/' || errfile);
   xmlparser.setPreserveWhiteSpace(p, TRUE);
   xmlparser.setBaseDir(p, dir);

-- parse xml file
   dbms_output.put_line('Parsing XML document ' || dir || '/' || xmlfile);
   xmlparser.parse(p, dir || '/' || xmlfile);

-- get document
   xmldoc := xmlparser.getDocument(p);

-- parse xsl file
   dbms_output.put_line('Parsing XSL document ' || dir || '/' || xslfile);
   xmlparser.parse(p, dir || '/' || xslfile);

-- get document
   xsldoc := xmlparser.getDocument(p);

   xslelem := xmldom.getDocumentElement(xsldoc);
   nspace := xmldom.getNamespace(xslelem);

-- print out some information about the stylesheet
   dbms_output.put_line('XSL Root element information');
   dbms_output.put_line('Qualified Name: ' ||
                        xmldom.getQualifiedName(xslelem));
   dbms_output.put_line('Local Name: ' ||
                        xmldom.getLocalName(xslelem));
   dbms_output.put_line('Namespace: ' || nspace);
   dbms_output.put_line('Expanded Name: ' ||
                        xmldom.getExpandedName(xslelem));

   xslcmds := xmldom.getChildrenByTagName(xslelem, '*', nspace);
   dbms_output.put_line('A total of ' || xmldom.getLength(xslcmds) ||
                       ' XSL instructions were found in the stylesheet');
-- make stylesheet
   ss := xslprocessor.newStylesheet(xsldoc, dir || '/' || xslfile);

-- process xsl
   proc := xslprocessor.newProcessor;
   xslprocessor.showWarnings(proc, true);
```

```
   xslprocessor.setErrorLog(proc, dir || '/' || errfile);

   dbms_output.put_line('Processing XSL stylesheet');
   docfrag := xslprocessor.processXSL(proc, ss, xmldoc);
   docfragnode := xmldom.makeNode(docfrag);

   dbms_output.put_line('Writing transformed document');
  xmldom.writeToFile(docfragnode, dir || '/' || resfile);

-- deal with exceptions
exception

when xmldom.INDEX_SIZE_ERR then
   raise_application_error(-20120, 'Index Size error');

when xmldom.DOMSTRING_SIZE_ERR then
   raise_application_error(-20120, 'String Size error');

when xmldom.HIERARCHY_REQUEST_ERR then
   raise_application_error(-20120, 'Hierarchy request error');

when xmldom.WRONG_DOCUMENT_ERR then
   raise_application_error(-20120, 'Wrong doc error');

when xmldom.INVALID_CHARACTER_ERR then
   raise_application_error(-20120, 'Invalid Char error');

when xmldom.NO_DATA_ALLOWED_ERR then
   raise_application_error(-20120, 'Nod data allowed error');

when xmldom.NO_MODIFICATION_ALLOWED_ERR then
   raise_application_error(-20120, 'No mod allowed error');

when xmldom.NOT_FOUND_ERR then
   raise_application_error(-20120, 'Not found error');

when xmldom.NOT_SUPPORTED_ERR then
   raise_application_error(-20120, 'Not supported error');

when xmldom.INUSE_ATTRIBUTE_ERR then
   raise_application_error(-20120, 'In use attr error');

end xslsample;
/
show errors;
```

Frequently Asked Questions About the XML Parser for PL/SQL

Why Do I Get an "Exception in Thread" Parser Error?

When I try to use the `oraxsl` I get the following: Exception in thread `main`:

```
java.lang.NoClassDefFoundError" oracle/xml/parser/v2/oraxsl.
```

How do I fix this?

Answer: If you are running outside the database you need to make sure the `xmlparserv2.jar` is explicitly in your `CLASS_PATH`, not simply its directory. If from the database you need to make sure it has been properly loaded and that JServer initialized.

How Do I Use the xmldom.GetNodeValue in PL/SQL?

I cannot get the element value using the PL/SQL XMLDOM. Here is the code fragment:

```
...nl := xmldom.getElementsByTagName(doc, '*');
len := xmldom.getLength(nl)
;-- loop through elements
   for i in 0..len-1 loop      n := xmldom.item(nl, i);
     elename := xmldom.getNodeName(n);
eleval := xmldom.getNodeValue(n);
...elename is Ok, but eleval is NULL.
```

Associating with a text node does not seem to work, or I am not doing it correctly? I receive a compile error, as in this example:

```
...t xmldom.DOMText;
...t := xmldom.makeText(n);
eleval := xmldom.getNodeValue(t);
```

What am I doing wrong?

Answer: To get the text node value associated with the element node, you must perform additional node navigation through `xmldom.getFirstChild(n)`.

To illustrate, change `printElements()` in `DOMSample.sql` as follows:

```
begin
-- get all elements
nl := xmldom.getElementsByTagName(doc, '*');
 len := xmldom.getLength(nl);
```

```
  -- loop through elements
for i in 0..len-1 loop      n := xmldom.item(nl, i);
      dbms_output.put(xmldom.getNodeName(n));
      -- get the text node associated with the element node
      n := xmldom.getFirstChild(n);
     if xmldom.getNodeType(n) = xmldom.TEXT_NODE then
dbms_output.put('=' &#0124; &#0124; xmldom.getNodeValue(n));
      end if;
     dbms_output.put(' ');
   end loop;
   dbms_output.put_line('');
end printElements;
```

This produces the following output, listing the elements:

```
family member=Sarah member=Bob member=Joanne member=Jim
```

The attributes of each element are:

```
family:familylastname val=Smithmember:membermemberid val=m1member:membermemberid
val=m2member:membermemberid val=m3 mom val=m1 dad val=m2member:membermemberid
val=m4 mom val=m1 dad val=m2
```

Can I Run the XDK for PL/SQL in an IIS Environment?

I downloaded XDK for PL/SQL but it requires OAS. Do you have any idea how to run this in an IIS environment?

Answer: If you're going to use IIS, it would be better to use the XML Parser for Java version 2. You'll also need Oracle9*i*.

How Do I Parse a DTD Contained in a CLOB with the XML Parser for PL/SQL?

I am having problems parsing a DTD file contained in a CLOB. I used the `xmlparser.parseDTDClob` API, provided by the XML Parser for PL/SQL.

I received the following error:

```
"ORA-29531: no method parseDTD in class oracle/xml/parser/plsql/XMLParserCover".
```

The procedure `xmlparser.parseDTDClob` calls a Java Stored Procedure `xmlparsercover.parseDTDClob`, which in turn calls another Java Stored Procedure `xmlparsercover.parseDTD`.

I have confirmed that the class file, `oracle.xml.parser.plsql.XMLParserCover`, has been loaded into the

database, and that it has been published. So the error message does not make sense. The procedure used to call `xmlparser.parseDTDClob` is:

```
create or replace procedure parse_my_dtd as p xmlparser.parser; l_clob clob;
begin   p := xmlparser.newParser;   select content into l_clob from
dca_documents where doc_id = 1;
xmlparser.parseDTDClob(p,l_clob,'site_template'); end; API Documentation for
xmlparser.parseDTDClob:

parseDTDClob PURPOSE   Parses the DTD stored in the given clob SYNTAX
PROCEDURE parseDTDClob(p Parser, dtd CLOB, root VARCHAR2); PARAMETERS   p
(IN)-  parser instance  dtd      (IN)-  dtd clob to parse  root     (IN)-  name
of the root element RETURNS   Nothing COMMENTS
```

Any changes to the default parser behavior should be made before calling this procedure. An application error is raised if parsing failed, for some reason. Description of the table `dca_documents`:

```
DOC_ID          NOT NULL   NUMBER  DOC_NAME       NOT NULL   VARCHAR2(350)
DOC_TYPE                   VARCHAR2(30)
 DESCRIPTION                VARCHAR2(4000)  MIME_TYPE
VARCHAR2(48)  CONTENT        NOT NULL   CLOB  CREATED_BY     NOT NULL
VARCHAR2(30)  CREATED_ON     NOT NULL   DATE  UPDATED_BY     NOT NULL
VARCHAR2(30)  UPDATED_ON     NOT NULL   DATE
```

The contents of the DTD:

```
<!ELEMENT site_template (component*)> <!ATTLIST site_template template_id CDATA
#REQUIRED> <!ATTLIST site_template template_name CDATA #REQUIRED> <!ELEMENT
component (#PCDATA)> <!ATTLIST component component_id ID #REQUIRED> <!ATTLIST
component parent_id ID #REQUIRED> <!ATTLIST component component_name ID
#REQUIRED>
```

Answer: This is a known issue in release 1.0.1 of the XML Parser for PL/SQL. Here is the workaround.

First, make a backup of
`./plsqlxmlparser_1.0.1/lib/sql/xmlparsercover.sql`

Then, in line 18 of `xmlparsercover.sql`, change the string
`oracle.xml.parser.plsql.XMLParserCover.parseDTD` to
`oracle.xml.parser.plsql.XMLParserCover.parseDTDClob`

Verify that Line 18 now reads:

```
procedure parseDTDClob(id varchar2, DTD CLOB, root varchar2, err in out
```

```
varchar2)    is language java name
'oracle.xml.parser.plsql.XMLParserCover.parseDTDClob(java.lang.String,
oracle.sql.CLOB, java.lang.String, java.lang.String[])';
```

Save the file, then rerun `xmlparsercover.sql` in SQL*Plus. Assuming you've loaded XMLParser version 2 release 2.0.2.6 into the database, this should solve your problem.

How Do I Use Local Variables with the XML Parser for PL/SQL?

I have just started using XML Parser for PL/SQL. I am have trouble getting the text between the begin tag and the end tag into a local variable. Do you have examples?

Answer: You just have to use the following:

```
selectSingleNode("pattern");
getNodeValue()
```

Remember, if you are trying to get value from a Element node, you have to move down to the `#text` child node, for example, `getFirstChild.getNodeValue()`

Suppose you need to get the text contained between the starting and ending tags of a `xmldom.DOMNode n`. The following two lines will suffice.

```
n_child:=xmldom.getFirstChild(n);
text_value:=xmldom.getNodeValue(n_child));
```

`n_child` is of type `xmldom.DOMNode`.

`text_value` is of type `varchar2`.

Why Do I Get a Security Error When I Grant JavaSysPriv to a User?

We are using the XML Parser for PLSQL and are trying to parse an XML document. We are getting a Java security error:

```
ORA-29532: Java call terminated by uncaught Java exception:
java.lang.SecurityException ORA-06512: at "NSEC.XMLPARSERCOVER", line 0
ORA-06512: at "NSEC.XMLPARSER", line 79 ORA-06512: at "NSEC.TEST1_XML line 36
ORA-06512: at line 5
```

Do we need to grant to user? The syntax appears correct. We also get the error when we run the demo.

Answer: If the document you are parsing contains a doctype which has a System URI with a protocol like `file:///` or `http:///` then you need to grant an appropriate privilege to your current database user to be able to "reach out of the database", so to speak, and open a stream on the file or `URL.CONNECT SYSTEM/MANAGER`. The following code should do it:

```
GRANT JAVAUSERPRIV, JAVASYSPRIV TO  youruser;
```

How Do I Install the XML Parser for PL/SQL with the JServer (JVM) Option?

I have downloaded and installed the `plxmlparser_V1_0_1.tar.gz`. The readme said to use `loadjava` to upload `xmlparserv2.jar` and `plsql.jar` in order. I tried to load `xmlparserv2.jar` using the following command:

```
loadjava -user test/test -r -v xmlparserv2.jar
```

to upload the jar file into Oracle8*i*. After much of the uploading, I got the following error messages:

```
identical: oracle/xml/parser/v2/XMLConstants is unchanged from previously loaded
fileidentical: org/xml/sax/Locator is unchanged from previously loaded
fileloading  : META-INF/MANIFEST.MFcreating : META-INF/MANIFEST.MFError while
creating resource META-INF/MANIFEST.MF    ORA-29547: Java system class not
available: oracle/aurora/rdbms/Compilerloading  :
oracle/xml/parser/v2/mesg/XMLErrorMesg_en_US.propertiescreating :
oracle/xml/parser/v2/mesg/XMLErrorMesg_en_US.propertiesError while creating
...
```

Then I removed -r from the previous command:

```
loadjava -user test/test -v xmlparserv2.jar
```

I still got errors but it's down to four:

```
.identical: org/xml/sax/Locator is unchanged from previously loaded fileloading
: META-INF/MANIFEST.MFcreating : META-INF/MANIFEST.MFError while creating
...
```

I think I have installed the JServer on the database correctly.

Answer: The JServer option is not properly installed if you're getting errors like this during `loadjava`. You need to run `INITJVM.SQL` and `INITDBJ.SQL` to get the JavaVM properly installed. Usually these are in the `./javavm` subdirectory of your Oracle Home.

How Do I Use the domsample Included with XML Parser for PL/SQL?

I am trying to execute `domsample` on `dom1151`. This is an example that is provided with the XML Parser for PL/SQL. The XML file `family.xml` is present in the directory `/hrwork/log/pqpd115CM/out`.

I am getting the following error:

```
Usage of domsample is domsample(dir, inpfile, errfile)

    SQL>
    begin
    domsample('/hrwork/log/pqpd115CM/out','family.xml','errors.txt');
      end;
      /
    Error generated :
    begin
    *
    ERROR at line 1:
    ORA-20100: Error occurred while parsing: No such file or directory
    ORA-06512: at "APPS.XMLPARSER", line 22
    ORA-06512: at "APPS.XMLPARSER", line 69
    ORA-06512: at "APPS.DOMSAMPLE", line 80
    ORA-06512: at line 2
```

Answer: From your description it sounds like you have not completed all of the steps in the sample and Readme without errors. After confirming that the `xmlparserv2.jar` is loaded, carefully complete the steps again.

How Do I Extract Part of a CLOB?

In an Oracle8*i* database, we have CLOBs which contain well-formed XML documents up to 1 MB in size.

We want the ability to extract only part of the CLOB (XML document), modify it, and replace it back in the database rather than processing the entire document.

Second, we want this process to run entirely on the database tier.

Which products or tools are needed for this? This may be possible with the JVM which comes with Oracle9*i*. There also may be some PL/SQL tools available to achieve this by means of stored procedures.

Answer: You can do this by using either of the following:

- Oracle XML Parser for PLSQL

- Create your own custom Java stored procedure wrappers over some code you write yourselves with the Oracle XML Parser for Java.

XML Parser for PLSQL has methods such as the following:

- `xmlparser.parseCLOB()`
- `xslProcessor.selectNodes()`, to find what part of the doc you are looking for
- `xmldom.*` methods to manipulate the content of the XML document
- `xmldom.writeToCLOB()` to write it back

If you wanted to do fine-detail updates on the text of the CLOB, you would have to use `DBMS_LOB.*` routines, but this would be tricky unless the changes being made to the content don't involve any increase or decrease in the number of characters.

Why Do I Get "Out of Memory" Errors in the XML Parser?

We are parsing a 50Mb XML file. We have upped the java_pool_size to 150Mb with a shared_pool_size of 200Mb. We get the following "out of memory" errors in the Oracle XML parser:

```
last entry at 2000-04-26 10:59:27.042:
VisiBroker for Java runtime caught exception:
java.lang.OutOfMemoryError
   at oracle.xml.parser.v2.XMLAttrList.put(XMLAttrList.java:251)
   at oracle.xml.parser.v2.XMLElement.setAttribute(XMLElement.java:260)
   at oracle.xml.parser.v2.XMLElement.setAttribute(XMLElement.java:228)
   at cars.XMLServer.processEXL(XMLServer.java:122)
```

It's trying to create a new XML attribute and crashes with `OutOfMemoryError`.

Answer: You should not be using the DOM parser for parsing a 50Mb XML file. You need to use the SAX parser, which parses files of arbitrary size because it does not create an in-memory tree of nodes as it goes.

If you are using DOM, you should seriously consider moving to SAX which processes the XML file sequentially instead of trying to build an in-memory tree that represents the file.

Using SAX we process XML files in excess of 180Mb without any problems and with very low memory requirements.

Rule of thumb for choosing between DOM and SAX:

DOM:

- DOM is very good when you need some sort of random access.
- DOM consumes more memory.
- DOM is also good when you are trying to transformations of some sort.
- DOM is also good when you want to have tree iteration and want to walk through the entire document tree.
- See if you can use more attributes over elements in your XML (to reduce the pipe size).

SAX:

- SAX is good when data comes in a streaming manner (using some input stream).

What Are the Memory Requirements for Using the PL/SQL Parser?

Answer: While the memory use is directly dependent on the document size, it should also be realized that the PL/SQL parser uses the Java parser and thus the Oracle JServer is being run. JServer typically requires 40-60 MB depending on its configuration.

Is JServer (JVM) Needed to Run XML Parser for PL/SQL?

Answer: Yes, if you are running the parser in the database, you do need JServer because the PL/SQL parser currently uses the XML Parser for Java under the covers. JServer exists in both the Standard and Enterprise versions. A forthcoming version of XML Parser for PL/SQL using C underneath is being developed for applications that do not have access to a Java Virtual Machine (JVM).

Frequently Asked Questions About Using the DOM API

What Does the XML Parser for PL/SQL Do?

Answer: The XML parser accepts any XML document and gives you a tree-based API (DOM) to access or modify the document's elements and attributes. It also supports XSLT which allows transformation from one XML document to another.

Can I Dynamically Set the Encoding in the XML Document?

Answer: No, you need to include the proper encoding declaration in your document according to the specification. You cannot use `setCharset(DOMDocument)` to set the encoding for the input of your document. `SetCharset(DOMDocument)` is used with `oracle.xml.parser.v2.XMLDocument` to set the correct encoding for the printing.

How Do I Get the Number of Elements in a Particular Tag?

How do I get the number of elements in a tag using the Parser?

Answer: You can use the `getElementByTagName (elem DOMElement, name IN VARCHAR2)` method that returns a `DOMNodeList` of all descent elements with a given tag name. You can then find out the number of elements in that `DOMNodeList` to determine the number of the elements in the particular tag.

How Do I Parse a String?

Answer: We do not currently have any method that can directly parse an XML document contained within a string. You can use one of the following as a workaround:

- function `parse (Parser, VARCHAR2)` to parse XML data stored in the given URL or the given file,
- function `parseBuffer (Parser, VARCHAR2)` to parser XML data stored in the given buffer, or
- function `parseCLOB (Parser, VARCHAR2)` to parse XML data stored in the give CLOB.

How Do I Display My XML Document?

Answer: If you are using Internet Explorer 5 as your browser, you can display the XML document directly. Otherwise, you can use our XSLT processor in version 2 of the parser to create the HTML document using an XSL Stylesheet. Our Java Transviewer bean also enables you to view your XML document.

How Do I Write the XML Data Back Using Special Character Sets?

Answer: You can specify the character sets for writing to a file or a buffer. Writing to a CLOB will be use the default character set for the database that you are writing to. Here are the methods to use:

- `procedure writeToFile(doc DOMDocument, fileName VARCHAR2, charset VARCHAR2);`
- `procedure writeToBuffer(doc DOMDocument, buffer IN OUT VARCHAR2, charset VARCHAR2);`
- `procedure writeToClob(doc DOMDocument, cl IN OUT CLOB, charset VARCHAR2);`

How Do I Obtain an Ampersand from Character Data?

Answer: You cannot have "raw" ampersands in XML data. You need to use the entity, `&` instead. This is defined in the XML standard.

How Do I Generate a Document Object from a File?

Answer: Refer to the following example:

```
inpPath VARCHAR2;
inpFile VARCHAR2;
p xmlparser.parser;
doc xmldom.DOMDocument;

-- initialize a new  parser object;
p := xmlparser.newParser;
-- parse the file
xmlparser.parse(p, inpPath || inpFile);
-- generate a document object
doc := xmlparser.getDocument(p);
```

Can the Parser Run on Linux?

Answer: As long as a version 1.1.x or 1.2.x JavaVM for Linux exists in your installation, you can run the Oracle XML Parser for Java there. Otherwise, you can use the C or C++ XML Parser for Linux.

Is Support for Namespaces and Schema Included?

Answer: The current XML Parsers support Namespaces. Schema support will be included in a future release.

Why Doesn't My Parser Find the DTD File?

Answer: The DTD file defined in the `<!DOCTYPE>` declaration must be relative to the location of the input XML document. Otherwise, you'll need to use the `setBaseDir(Parser, VARCHAR2)` functions to set the base URL to resolve the relative address of the DTD.

Can I Validate an XML File Using an External DTD?

Answer: You need to include a reference to the applicable DTD in your XML document. Without it there is no way that the parser knows what to validate against. Including the reference is the XML standard way of specifying an external DTD. Otherwise you need to embed the DTD in your XML Document.

Does the Parser Have DTD Caching?

Answer: Yes, DTD caching is optional and it is not enabled automatically.

How Do I Get the DOCTYPE Tag into the XML Document After It Is Parsed?

Answer: You need to do some preprocessing to the file, and then put it through the DOM parser again, which will produce a valid, well-formed XML document with the `DOCTYPE` tag contained within.

How Does the XML DOM Parser Work?

Answer: The parser accepts an XML formatted document and constructs in memory a DOM tree based on its structure. It will then check whether the document is well-formed and optionally whether it complies with a DTD. It also provides methods to traverse the tree and return data from it.

How Do I Create a Node Whose Value I Can Set Later?

Answer: If you check the DOM spec referring to the table discussing the node type, you will find that if you are creating an element node, its `nodeValue` is to be null and hence cannot be set. However, you can create a text node and append it to the element node. You can store the value in the text node.

How Do I Extract Elements from the XML File?

Answer: If you're using DOM, the you can use the `NamedNodeMap` methods to get the elements.

How Do I Append a Text Node to a DOMElement Using PL/SQL Parser?

Answer: Use the `createTextNode()` method to create a new text node. Then convert the `DOMElement` to a `DOMNode` using `makeNode()`. Now, you can use `appendChild()` to append the text node to the `DOMElement`.

I Am Using XML Parser with DOM; Why Can I Not Get the Actual Data?

Answer: You need to check at which level your data resides. For example,

- `<?xml version=1.0 ?>`
- `<greeting>Hello World!</greeting>`

The text is the first child node of the first DOM element in the document. According to the DOM Level 1 spec, the value of an `ELEMENT` node is null and the `getNodeValue()` method will always return null for an `ELEMENT` type node. You have to get the `TEXT` children of an element and then use the `getNodeValue()` method to retrieve the actual text from the nodes.

Can the XML Parser for PL/SQL Produce Non-XML Documents?

Answer: Yes it can.

I Cannot Run the Sample File. Did I Do Something Wrong In the Installation?

Answer: Here are two frequently missing steps in installing the PL/SQL parser:

- initialize the JServer -- run `$ORACLE_HOME/javavm/install/initjvm.sql`
- load the included jar files from the parser archive.

How Do I Parse a DTD in a CLOB?

I am having problems parsing a DTD file contained in a CLOB. I used the `xmlparser.parseDTDClob` API, provided by the XML Parser for PL/SQL.

The following error was thrown:

```
"ORA-29531: no method parseDTD in class oracle/xml/parser/plsql/XMLParserCover"
```

I managed to work out the following:

The procedure `xmlparser.parseDTDClob` calls a Java Stored Procedure `xmlparsercover.parseDTDClob`, which in turn calls another Java Stored Procedure `xmlparsercover.parseDTD`.

I have confirmed that the class file `oracle.xml.parser.plsql.XMLParserCover` has been loaded into the database, and that it has been published. So the error message does not make sense.

I am not able to figure out whether I am doing it right or whether this is a bug in the parser API.

```
The procedure use to call "xmlparser.parseDTDClob" :
-----------------------------------------------------
create or replace procedure parse_my_dtd as
p xmlparser.parser;
l_clob clob;
begin
  p := xmlparser.newParser;
  select content into l_clob from dca_documents where doc_id = 1;
  xmlparser.parseDTDClob(p,l_clob,'site_template');
end;
```

API Documentation for `xmlparser.parseDTDClob`:

```
parseDTDClob
PURPOSE
  Parses the DTD stored in the given clob
SYNTAX
  PROCEDURE parseDTDClob(p Parser, dtd CLOB, root VARCHAR2);
PARAMETERS
  p        (IN)-  parser instance
 dtd      (IN)-  dtd clob to parse
 root     (IN)-  name of the root element
RETURNS
  Nothing
COMMENTS
```

Any changes to the default parser behavior should be effected before calling this procedure. An application error is raised if parsing failed, for some reason.

Description of the table `dca_documents`:

```
DOC_ID          NOT NULL    NUMBER
DOC_NAME        NOT NULL    VARCHAR2(350)
DOC_TYPE                    VARCHAR2(30)
DESCRIPTION                 VARCHAR2(4000)
MIME_TYPE                   VARCHAR2(48)
CONTENT         NOT NULL    CLOB
CREATED_BY      NOT NULL    VARCHAR2(30)
CREATED_ON      NOT NULL    DATE
UPDATED_BY      NOT NULL    VARCHAR2(30)
UPDATED_ON      NOT NULL    DATE
```

The contents of the DTD:

```
<!ELEMENT site_template (component*)>
<!ATTLIST site_template template_id CDATA #REQUIRED>
<!ATTLIST site_template template_name CDATA #REQUIRED>
<!ELEMENT component (#PCDATA)>
<!ATTLIST component component_id ID #REQUIRED>
<!ATTLIST component parent_id ID #REQUIRED>
<!ATTLIST component component_name ID #REQUIRED>
```

Answer 1: It appears to be a typo in the `xmlparsercover.sql` script which is defining the Java Stored Procedures that wrap the XMLParser. It mentions the Java method name `parseDTD` in the 'is language java name' part when `parseDTD` should be `parseDTDClob` (case-sensitive).

If you:

1. Make a backup copy of this script
2. Edit the line that reads:

   ```
   procedure parseDTDClob(id varchar2,
   dtd CLOB, root varchar2, err in out varchar2) is language java name
   'oracle.xml.parser.plsql.XMLParserCover.parseDTD (java.lang.String,
   oracle.sql.CLOB, java.lang.String, java.lang.String[])';
   ```

   to say:

   ```
   procedure parseDTDClob(id varchar2,
   dtd CLOB, root varchar2, err in out varchar2) is language java name
   'oracle.xml.parser.plsql.XMLParserCover.parseDTDClob
   (java.lang.String, oracle.sql.CLOB, java.lang.String,
   java.lang.String[])';
   ```

   That is, change the string:

'oracle.xml.parser.plsql.XMLParserCover.parseDTD

to

'oracle.xml.parser.plsql.XMLParserCover.parseDTDClob

and rerun the `xmlparsercover.sql` script you should be in business.

I filed a bug 1147031 to get this typo corrected in a future release.

Note: Your DTD had syntactic errors in it, but I was able to run the following without problem after making the change:

```
declare
   c clob;
   v varchar2(400) :=
'<!ELEMENT site_template  (component* )>
<!ATTLIST site_template  template_name CDATA  #IMPLIED
                         tempmlate_id  CDATA  #IMPLIED >
<!ELEMENT component  (#PCDATA )>
<!ATTLIST component  component_id   ID     #REQUIRED
                     parent_id      IDREF  #IMPLIED
                     component_name CDATA  #IMPLIED >';
begin
   delete from dca_documents;
   insert into dca_documents values(1,empty_clob())
    returning content into c;
   dbms_lob.writeappend(c,length(v),v);
   commit;
   parse_my_dtd;
end;
```

Answer 2: What do you want to do with the LOB? The LOB can either be a temporary LOB or a persistent LOB. In case of persistent LOBs, you need to insert the value into a table. In case of temp LOB you can instantiate it in your program.

For example:

```
   persistant lob
   declare
     clob_var CLOB;
   begin
     insert into tab_xxx values(EMPTY_CLOB()) RETURNING clob_col INTO
clob_var;
      dbms_lob.write(,,,,);
    // send to AQ
   end;
   temp lob -----
```

```
declare
  a clob;
begin
   dbms_lob.createtemporary(a,DBMS_LOB.SESSION);
   dbms_lob.write(...);
  // send to AQ

end;
/
```

Also refer to *Oracle9i Application Developer's Guide - Large Objects (LOBs)*. There are six books (in PDF), one for each language access (C(OCI), Java, PL/SQL, Visual Basic, Pro*C/C++, Pro*Cobol)) and it is quite comprehensive. If this is PL/SQL, I believe you can just do the following:

```
myClob CLOB = clob();
```

I have tried the `DBMS_LOB.createtemporary()` which works.

Answer 3: Here's what you need to do if you are using LOBs with AQ:

1. Create an ADT with one of the fields of type CLOB.

   ```
   create type myAdt (id NUMBER, cdata CLOB);
   ```

   The queue table must be declared to be of type myAdt

2. Instantiate the object - use `empty_clob()` to fill the LOB field

   ```
   myMessage := myAdt(10, EMPTY_CLOB();
   ```

3. Enqueue the message

   ```
   clob_loc clob;
   enq_msgid RAW(16);
   DBMS_AQ.enqueue('queue1', enq_opt, msg_prop, myMessage, enq_msgid)
   ```

4. Get the LOB locator

   ```
   select t.user_data.cdata into clob_loc
   from qtable t  where t.msgid
   = enq_msgid;
   ```

5. Populate the CLOB using `dbms_lob.write`

6. Commit

There is an example of this is in the *Oracle9i Application Developer's Guide - Advanced Queuing*. If you are using the Java API for AQ, the procedure is slightly more complicated.

Why Do I Get Errors When Parsing a Document?

I downloaded the javaparser, version 2 and the XML parser utility, and I'm using the PL/SQL parser interface. I have an XML file that is a composite of three tags and when parsing it generates the following error:

```
ORA-20100: Error occurred while parsing: Unterminated string
```

When I separate the document into individual tags, two are OK, but the third generates this error:

```
ORA-20100: Error occurred while parsing: Invalid UTF8 encoding
```

1. Why is the error different when separating the data?
2. I have not been able to find an "unterminated string" in the document.
3. I'm fairly anxious since this is the only way the data is coming and I don't have time to figure out another parser.

Answer: If you document is the "composite of three tags" then it is not a well-formed document as it has more than one root element. Try putting a start and end tag around the three.

How Do I Use PLXML to Parse a Given URL?

I am working with the XML parser for PL/SQL on NT. According to your Parser API documentation it is possible to parse a given URL, too:> Parses XML stored in the given URL/file and returns> the built DOM DocumentNow, parsing from file works fine, but any form of URL raises `ORA-29532:... java.io.FileNotFoundException`.

Can you give an example of a call?

Answer: To access external URLs, you need set up your proxy host and port. For example using this type of syntax:

```
java -Dhttp.proxyHost=myproxy.mydomain.com -Dhttp.proxyPort=3182DOMSample myxml.xml
```

How Do I Use the XML Parser to Parse HTML?

We need to parse HTML files as follows:

1. Find each `a href`
2. For each `a href` found, extract the file/pathname being linked to

3. Substitute a database procedure call for the `a href`, passing the file/pathname as a parameter.

Does it make sense to use the PL/SQL XML parser to do this? If so, how easy/hard would it be, and how can we find out how to do this?

Answer: Since HTML files aren't necessary well formed XML documents, are you sure you want to use XML parser? Won't Perl be a better choice? I'm not sure whether PL/SQL parser supports the following methods but just for your information:

1. `getElementsByTagName()` retrieves all matching nodes.
2. `getNodeValue()` will return a string.
3. `setNodeValue()` sets node values.

Answer 3: It supports those methods, but not over an ill-formed HTML file.

How Do I Move Data to a Web Browser Using PL/SQL and Oracle 7.3.4?

I'm trying to get the data to a Web browser in the client side while all the processing has to take place on the server (Oracle 7 release 7.3.4), using:

- XML Parser for PL/SQL
- XSQL servlet

Are these two components sufficient to get the job done?

Answer: Dependencies for XSQL Page Processor states:

- Oracle XML Parser V2 R2.0.2.5
- Oracle XML-SQL Utility for Java
- Web server supporting Java Servlets
- JDBC driver

You'll also need XSQL Page Processor itself.

Does the XML Parser for Java Work with Oracle 7.3.4?

Does the XML Parser for Java version 2, work with Oracle 7 release 7.3.4.?

Is XML- SQL Utility part of XML Parser for Java version 2, or does it need to be downloaded separately?

Answer:

1. The XML Parser for Java version 2 works with Oracle 7 release 7.3.4 as long as you have the proper JDBC driver and run it in a VM on a middle tier or client.
2. The XML-SQL Utility includes a copy of the version 2 parser in its download, as it requires it.

getNodeValue(): Getting the Value of DomNode

I am having problems obtaining the value between XML tags after using `xmlparser()`. Below is code from the `DOMSAMPLE.SQL` example:

```
-- loop through elementsfor i in 0..len-1 loop   n := xmlparser.item(nl, i);
   dbms_output.put(xmlparser.getNodeName(n)
```

Answer: I encountered the same problem. I found out that `getNodeValue()` on Element Node returns null. However, `getNodeValue()` on the *text* node returns the value.

How Do I Retrieve All Children or Grandchildren of a Node?

Is there a way to retrieve all children or grandchildren, and so on, of a particular node in a DOM tree using the DOM API? Or is there a work-around? We are using the XML Parser for PL/SQL.

Answer: Try the following:

```
DECLARE  nodeList   xmldom.DOMNodeList;
theElement xmldom.DOMElement;
BEGIN    :nodeList := xmldom.getElementsByTagName( theElement,'*');
:END;
```

This gets all children nodes rooted as the element in "theElement".

What Causes ora-29532 "Uncaught java exception:java.lang.ClassCastException?"

We want to parse XML, apply XSL, and get the transformed result in the form of an XML document. We are using XML Parser for PL/SQL. Our script does not add PI instruction `<?xml version="1.0"?>` to the transformed result.

`XSLProcessor.processXSL` method returns `documentfragment` object.

Create `DOMdocument` object from that `documentfragment` object using:
`finaldoc := xmldom.MakeDocument(docfragnode);`

Write to result file using where `finaldoc` is created of type `xmldom.DOMDocument:`

```
xmldom.writeToFile(finaldoc, dir || '/' || resfile);
```

This method is available for `DOMDocument`, but we are getting:

```
ora-29532 "Uncaught java exception:java.lang.ClassCastException"
```

I am not sure if converting `documentfragment` to `domdocument` object adds instruction `"<?xml version="1.0"?> "`, or must we add this instruction through XSL?

Answer: If you have created a new `DOMDocument` and then appended the document fragment to it, then you can use `xmldom.WriteToBuffer()` or similar routine to serialize with the XML declaration in place.

21

XSLT Processor for PL/SQL

This chapter contains the following sections:

Using the XML Parser for PL/SQL: XSLT Processor (DOM Interface)

Extensible Stylesheet Language Transformation, abbreviated XSLT (or XSL-T), describes rules for transforming a source tree into a result tree. A transformation expressed in XSLT is called a stylesheet.

The transformation specified is achieved by associating patterns with templates defined in the stylesheet. A template is instantiated to create part of the result tree.

This PLSQL implementation of the XSL processor follows the W3C XSLT working draft (rev WD-xslt-19990813) and includes the required behavior of an XSL processor in terms of how it must read XSLT stylesheets and the transformations it must effect.

The types and methods described in this document are made available by the PLSQL package, `xslprocessor()`.

Figure 21–1 shows the XML Parser for PL/SQL XSLT Processor main functionality.

1. The Process Stylesheet process receives input from the XML document and the selected Stylesheet, which may or may not be indicated in the XML document. Both the stylesheet and XML document can be the following types:

   - File name
   - Varchar buffer
   - CLOB

   The XML document can be input 1 through n times.

2. The parsed XML document inputs `XSLProcessor.processXSL(xslstylesheet,xml instance)` procedure, where:

   - XML document is indicated in the "xml instance" argument
   - Stylesheet input is indicated in the "xslstylesheet" argument

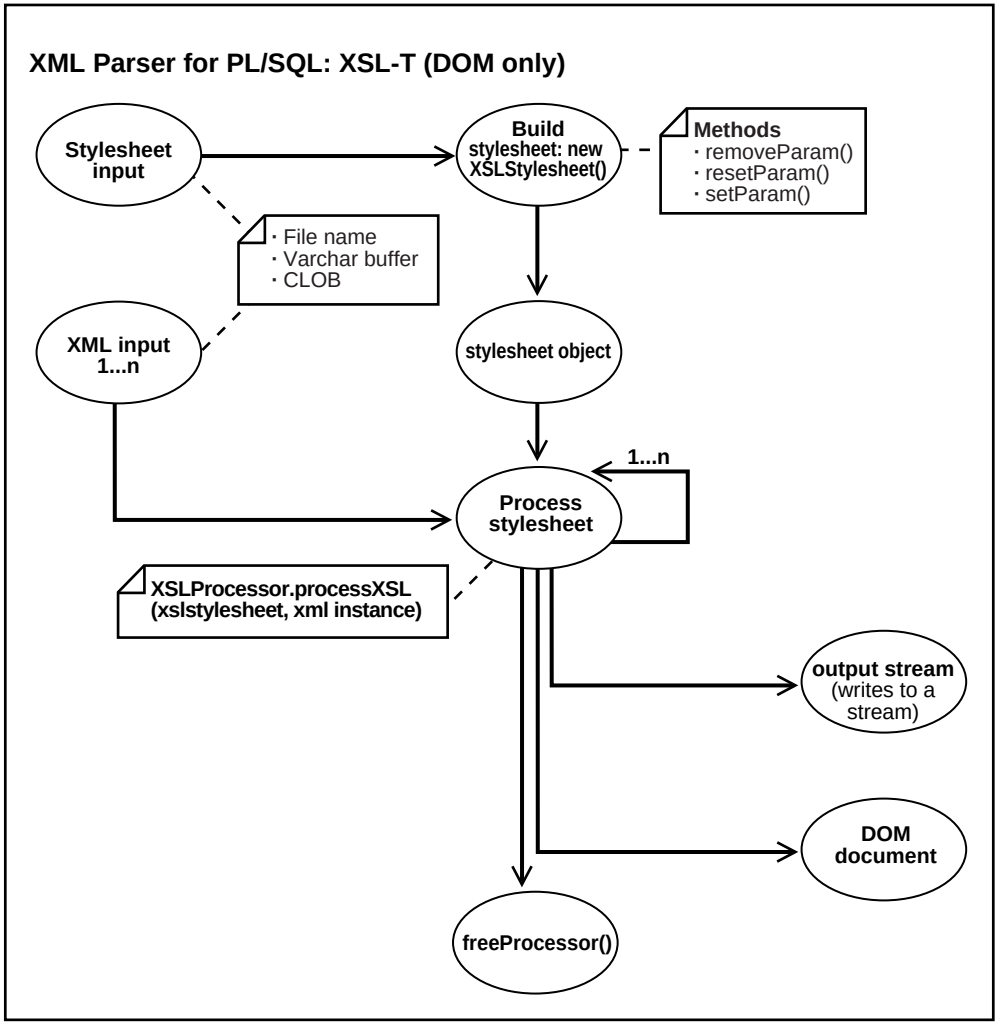
3. Build the stylesheet using the Stylesheet input to the `XSLStylesheet()` procedure. The following methods are available for this procedure:

   - `removeParam()`
   - `resetParam()`
   - `setParam()`

This produces a stylesheet object which then inputs the "Process Stylesheet" step using procedure, `XSLProcessor.processXSL(xslstylesheet,xml instance)`.

4. The "Process stylesheet" process can be repeated 1 through n times. In other words, the same stylesheet can be applied to multiple parsed XML documents to transform them wither into an XML document, HTML document, or other text based format.
5. The resulting parsed and transformed document is output either as a stream or a DOM document.
6. When the XSLT process if complete, call the freeProcessor() procedure to free up any temporary structures and the XSLProcessor procedures used in the XSL transformation process.

Figure 21–1 "XML Parser for PL/SQL: XSLT processor (DOM Interface)



XML Parser for PL/SQL: XSLT Processor — Default Behavior

The following is the default behavior for the XML Parser for PL/SQL XSLT Processor:

- A result tree which can be accessed by DOM APIs is built

- Errors are not recorded unless an error log is specified; however, an application error will be raised if parsing fails

XML Parser for PL/SQL Example: XSL — iden.xsl

This XSL file inputs the xslsample.sql.

```
<?xml version="1.0"?>

<!-- Identity transformation -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
 <xsl:template match="*|@*|comment()|processing-instruction()|text()">
  <xsl:copy>
  <xsl:apply-templates select="*|@*|comment()|processing-instruction()|text()"/>
  </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

22

XML Schema Processor for PL/SQL

This chapter contains the following sections:

Oracle XML Schema Processor for PL/SQL

The XML Schema Processor for Java is a component of the XDK that supports simple and complex datatypes in XML applications.

Building Server-Side XML Schema Validation

This chapter gives an introduction to the XML schema validation process using the XDK for Java and discusses how to build an Oracle Java Stored Procedure to perform the schema validation on the server-side of the Oracle Database. The included sample code also demonstrates the deployment procedure for Java Stored Procedures.

XML Schema Validation can provide a flexible and portable form of data validation for use in your applications. You can implement the XML validation process in your client-side or mid-tier applications, but if you want to have either:

- control of data validation whenever the data is updated or inserted, or
- use of the data management capability of the Oracle database,

then putting your data validation process inside a trigger or your PL/SQL procedures on the server-side is a good solution. Since there is not a builtin PL/SQL API to do XML Schema validation, we can create one using Java Stored Procedures.

The first step in building a Java Stored Procedure for XML Schema validation is to select the components and decide the environment requirements. The components you need are:

- XML Schema Processor for Java (xschema.jar])
- XML Parser for Java (xmlparserv2.jar)

Both of these are part of the Oracle XML Developer's Kit for Java. The Oracle database (8.1.6 version and above) is also needed because these versions fully support Java Stored Procedures.

If you download the XDK for Java and have an Oracle 8.1.6 database or above, you can follow the following steps to build the Java Stored Procedure and take advantage of XML Schema for data validation.

> **See Also:** Download XML Developers Kit for Java from the Oracle Technology Network:
>
> http://technet.oracle.com/tech/xml/xdkhome.html
>
> Source code for the demo is `xdksample_093001.zip`

Creating the Java Classes for XML Schema Validation

To build the Java Class for XML Schema Validation, two XDK packages, XML Schema Processor and XML Parser are needed:

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

To be able to accept the inputs from PL/SQL, we need another package:

```
import oracle.sql.CHAR;
```

You need to set `xmlparserv2.jar`, `xschema.jar` and `classes12.zip` in the `CLASSPATH`. The JDBC library `classes12.zip` is for JDK 1.2.x. If you are using JDK 1.1.x, `classes111.zip` is required.

The SchemaUtil Class is:

```
public class SchemaUtil
{

   public static String validation(CHAR xml, CHAR xsd)
   throws Exception
   {
      //Build Schema Object
      XSDBuilder builder = new XSDBuilder();
      byte [] docbytes = xsd.getBytes();
      ByteArrayInputStream in = new   ByteArrayInputStream(docbytes);
      XMLSchema schemadoc = (XMLSchema)builder.build(in,null);
      //Parse the input XML document with Schema Validation
      docbytes = xml.getBytes();
      in = new ByteArrayInputStream(docbytes);
      DOMParser dp  = new DOMParser();
      // Set Schema Object for Validation
      dp.setXMLSchema(schemadoc);
      dp.setValidationMode(XMLParser.SCHEMA_VALIDATION);
      dp.setPreserveWhitespace (true);
      StringWriter sw = new StringWriter();
      dp.setErrorStream (new PrintWriter(sw));
      try
      {
         dp.parse (in);
         sw.write("The input XML parsed without errors.\n");
      }
      catch (XMLParseException pe)
      {
```

```
            sw.write("Parser Exception: " + pe.getMessage());
          }
        catch (Exception e)
          {
            sw.write("NonParserException: " + e.getMessage());
          }
          return sw.toString();
    }
  }
```

This class defines a single method, `validation`, which does the XML Schema validation for the input XML document and returns the error messages.

To compile the class, use following command line:

```
javac SchemaUtil.java
```

This produces the compiled Java class, `SchemaUtil.class`.

Loading and Resolving the Java Class

With the utility `loadjava`, you can upload the Java source, class, and resource files into an Oracle database, where they are stored as Java schema objects. You can run `loadjava` from the command line or from an application, and you can specify several options, including a resolver. Make sure you have `$ORACLE_HOME\bin` in your System Path to be able to run `loadjava`.

Before loading the SchemUtil.class into the database, you need to check if the correct version of the two dependent XDK packages are loaded into the logon database schema (in this case `xdkdemo/xdkdemo`).

```
connect xdkdemo/xdkdemo
```

To check the status of the `oracle.xml.parser.v2.DOMParser` class, you can use the following SQL statement:

```
ELECT SUBSTR(dbms_java.longname(object_name),1,35) AS class, status
FROM   all_objects
WHERE  object_type = 'JAVA CLASS'
       AND object_name = dbms_java.shortname('oracle/xml/parser/v2/DOMParser');
```

If you see the result:

```
CLASS                                                    STATUS
-------------------------------------------------------------
oracle/xml/parser/v2/DOMParser                           VALID
```

then the Oracle XML Parser for Java is already installed and ready to be used.

If you see the preceding result, but the status is INVALID, try the command:

```
ALTER JAVA CLASS _oracle/xml/parser/v2/DOMParser Resolve
```

If the verification procedure produces the SQL*Plus message 'no rows selected', you need to load the XML Parser into the database by:

```
loadjava -resolve -verbose -user  xdktemp/xdktemp xmlparserv2.jar
```

If the parser is installed, then you do not need to complete any further installation steps. The SQL command for status checking will be:

```
SELECT SUBSTR(dbms_java.longname(object_name),1,35) AS class, status
FROM   all_objects
WHERE  object_type = 'JAVA CLASS'
        AND object_name =
dbms_java.shortname('oracle/xml/parser/schema/XMLSchema');
```

Before loading the SchemaUtil.class, make sure that the loaded XML Parser has the same version with which you compiled the SchemaUtil.class. The following code can be used to check the current version of the loaded Oracle XML Parser:

```
CREATE OR REPLACE FUNCTION XMLVersion RETURN VARCHAR2
IS LANGUAGE JAVA NAME
'oracle.xml.parser.v2.XMLParser.getReleaseVersion() returns java.lang.String';
/
CREATE OR REPLACE Procedure getXMLVersion AS
begin
       dbms_output.put_line(XMLVersion());
end;
/
```

Then by issuing the command:

```
SQL> set serveroutput on
SQL> exec getXMLVersion;
```

You should receive the following result:

```
Oracle XDK Java      9.0.2.0.0A      Beta
```

If the version does not match, you need to drop the package and reload it. To drop the package, you can issue the following command line:

```
dropjava -verbose -user xdktemp/xdktemp  xmlparserv2.jar xschema.jar
```

Once all of the versions are synchronizeded, you can finally load the SchemaUtil.class by:

```
loadjava -resolve -verbose -user  xdktemp/xdktemp SchemaUtil.class
```

Publishing the Java Class by Defining the Specification

For each Java method callable from SQL, you must write a call specification in Java, which exposes the method's top-level entry point to the Oracle server.

```
CREATE OR REPLACE FUNCTION SchemaValidation(xml IN VARCHAR2,xsd IN VARCHAR2)
return varchar2
IS LANGUAGE JAVA NAME
'SchemaUtil.validation(oracle.sql.CHAR,oracle.sql.CHAR) returns
      java.lang.String';
```

Now the Java stored procedure specification is created, both SQL and PL/SQL can call it as if it were a PL/SQL function.

Example Using the Stored Procedures

You can call Java stored procedures from SQL DML statements, PL/SQL blocks, and PL/SQL subprograms. Using the SQL CALL statement, you can also call them from the top level (from SQL*Plus, for example) and from database triggers. The following example shows how to do XML Schema Validation using the created Java stored procedure.

Creating a Database Schema to store XML and XML Schema Documents

```
create table schema_tab(id number, xsd VARCHAR2(4000));
create table xml_tab(id number, xml VARCHAR2(4000));
```

Loading the XML Schema Document into the Database

You can use the SQL commands to insert the data in DBData.sql:

```
INSERT INTO schema_tab(1, '[XML schema]');
```

Calling the Java Stored Procedure from the Trigger of the xml_tab Table

```
--Write XML Buffer to Output
CREATE OR REPLACE PROCEDURE printBufferOut(xmlstr IN OUT NOCOPY VARCHAR2) AS
BEGIN
```

```
   line   VARCHAR2(20000);
   nlpos  INTEGER;
   LOOP
     EXIT WHEN xmlstr is null;
     nlpos := instr(xmlstr,chr(10));
     line := substr(xmlstr,1,nlpos-1);
     -- print line
     IF(length(line) <250) THEN
         dbms_output.put_line('| '||line);
     ELSE
         dbms_output.put('| ');
         LOOP
            EXIT WHEN line is null;
            dbms_output.put_line(substr(line,1,250));
            line := substr(line,250+1);
         END loop;
     END if;
     xmlstr := substr(xmlstr,nlpos+1);
     IF (nlpos = 0) THEN
       dbms_output.put_line('| '||xmlstr);
       EXIT;
     END if;
   END LOOP;
END printBufferOut;
/

show errors;
CREATE OR REPLACE PROCEDURE dbvalid(xmlid IN NUMBER, xsdid IN NUMBER) IS
       p_xml varchar2(4000);
       p_xsd varchar2(4000);
       p_out varchar2(4000);
     begin
        select xml into p_xml from xml_tab where id=xmlid;
        select xsd into p_xsd from schema_tab where id=xsdid;
             p_out := SchemaValidation(p_xml,p_xsd);
        printBufferOut(p_out);
      end;
/
```

For the date with the `xdksample_093001.zip` you can execute the command and get the following result:

```
SQL> exec dbvalid(1,1);
| The input XML parsed without errors.
PL/SQL procedure successfully completed.
```

```
SQL> exec dbvalid(2,1);
| | <Line 5, Column 42>: XSD-2023: (Error) Invalid value of attribute:
'1999-11-31'
| <Line 21, Column 27>: XSD-2105: (Error) Identity constraint validation error:
'Key sequence not found in key reference'.
| | Parser Exception: Invalid value of attribute: '1999-11-31'
PL/SQL procedure successfully completed.
```

You can now use this Java Stored Procedure to validate the XML document using PL/SQL.

23

XSU for PL/SQL

This chapter contains the following sections:

See Also: Chapter 8, "XML SQL Utility (XSU)" for information about XSU in general.

XSU PL/SQL API

XML SQL Utility (XSU) PL/SQL API reflects the Java API in the generation and storage of XML documents from and to a database. `DBMS_XMLQuery` and `DBMS_XMLSave` are the two packages that reflect the functions in the Java classes - `OracleXMLQuery` and `OracleXMLSave`. Both of these packages have a context handle associated with them. Create a context by calling one of the constructor-like functions to get the handle and then use the handle in all subsequent calls.

XSU Supports XMLType

From Oracle9i Release 2 (9.2), XSU supports XMLType. Using XSU with XMLType is useful if, for example, you have XMLType columns in objects or tables.

> **See Also:** *Oracle9i XML Database Developer's Guide - Oracle XML DB*, in particular, the chapter on Generating XML, for examples on using XSU with XMLType.

Generating XML with DBMS_XMLQuery()

Generating XML results in a CLOB that contains the XML document. To use `DBMS_XMLQuery` and the XSU generation engine, follow these steps:

1. Create a context handle by calling the `DBMS_XMLQuery.getCtx` function and supplying it the query, either as a `CLOB` or a `VARCHAR2`.
2. Bind possible values to the query using the `DBMS_XMLQuery.bind` function. The binds work by binding a name to the position. For example, the query can be `select * from emp where empno = :EMPNO_VAR`. Here you are binding the value for the `EMPNO_VAR` using the `setBindValue` function.
3. Set optional arguments like the `ROW` tag name, the `ROWSET` tag name, or the number of rows to fetch, and so on.
4. Fetch the XML as a CLOB using the `getXML()` functions. `getXML()` can be called to generate the XML with or without a DTD or schema.
5. Close the context.

Here are some examples that use the `DBMS_XMLQuery` PL/SQL package.

XSU Generating XML Example 1: Generating XML from Simple Queries (PL/SQL)

In this example, you select rows from table `emp`, and obtain an XML document as a CLOB. First get the context handle by passing in a query and then call the

getXMLClob routine to get the CLOB value. The document is in the same encoding as the database character set.

```
declare
  queryCtx DBMS_XMLquery.ctxType;
  result CLOB;
begin

  -- set up the query context...!
  queryCtx := DBMS_XMLQuery.newContext('select * from emp');

  -- get the result..!
  result := DBMS_XMLQuery.getXML(queryCtx);
  -- Now you can use the result to put it in tables/send as messages..
  printClobOut(result);
  DBMS_XMLQuery.closeContext(queryCtx);  -- you must close the query handle..
end;
/
```

XSU Generating XML Example 2: Printing CLOB to Output Buffer

printClobOut*()* is a simple procedure that prints the CLOB to the output buffer. If you run this PL/SQL code in SQL*Plus, the result of the CLOB is printed to screen. Set the serveroutput to on in order to see the results.

```
CREATE OR REPLACE PROCEDURE printClobOut(result IN OUT NOCOPY CLOB) is
xmlstr varchar2(32767);
line varchar2(2000);
begin
  xmlstr := dbms_lob.SUBSTR(result,32767);
  loop
    exit when xmlstr is null;
    line := substr(xmlstr,1,instr(xmlstr,chr(10))-1);
    dbms_output.put_line('| '||line);
    xmlstr := substr(xmlstr,instr(xmlstr,chr(10))+1);
  end loop;
end;
/
```

XSU Generating XML Example 3: Changing ROW and ROWSET Tag Names

With the XSU PL/SQL API you can also change the ROW and the ROWSET tag names. These are the default names placed around each row of the result, and

round the whole document, respectively. The procedures, `setRowTagName` and `setRowSetTagName` accomplish this as shown in the following example:

```
--Setting the ROW tag names

declare
   queryCtx DBMS_XMLQuery.ctxType;
   result CLOB;
begin
   -- set the query context.
   queryCtx := DBMS_XMLQuery.newContext('select * from emp');

   DBMS_XMLQuery.setRowTag(queryCtx,'EMP'); -- sets the row tag name
   DBMS_XMLQuery.setRowSetTag(queryCtx,'EMPSET'); -- sets rowset tag name

   result := DBMS_XMLQuery.getXML(queryCtx); -- get the result

   printClobOut(result);  -- print the result..!
   DBMS_XMLQuery.closeContext(queryCtx);  -- close the query handle;
end;
/
```

The resulting XML document has an `EMPSET` document element. Each row is separated using the `EMP` tag.

XSU Generating XML Example 4: Using setMaxRows() and setSkipRows()

The results from the query generation can be paginated by using:

- `setMaxRows` function. This sets the maximum number of rows to be converted to XML. This is relative to the current row position from which the last result was generated.
- `setSkipRows` function. This specifies the number of rows to skip before converting the row values to XML.

For example, to skip the first 3 rows of the emp table and then print out the rest of the rows 10 at a time, you can set the `skipRows` to 3 for the first batch of 10 rows and then set `skipRows` to 0 for the rest of the batches.

As in the case of XML SQL Utility's Java API, call the `keepObjectOpen()` function to ensure that the state is maintained between fetches. The default behavior is to close the state after a fetch. For multiple fetches, you must determine when there are no more rows to fetch. This can be done by setting the

`setRaiseNoRowsException()`. This causes an exception to be raised if no rows are written to the CLOB. This can be caught and used as the termination condition.

```
-- Pagination of results

declare
  queryCtx DBMS_XMLquery.ctxType;
  result CLOB;
begin

  -- set up the query context...!
  queryCtx := DBMS_XMLQuery.newContext('select * from emp');

  DBMS_XMLQuery.setSkipRows(queryCtx,3); -- set the number of rows to skip
  DBMS_XMLQuery.setMaxRows(queryCtx,10); -- set the max number of rows per fetch

  result := DBMS_XMLQuery.getXML(queryCtx); -- get the first result..!

  printClobOut(result); -- print the result out.. This is you own routine..!
  DBMS_XMLQuery.setSkipRows(queryCtx,0); -- from now don't skip any more rows..!

  DBMS_XMLQuery.setRaiseNoRowsException(queryCtx,true);
                                        -- raise no rows exception..!
  begin
    loop  -- loop forever..!
      result := DBMS_XMLQuery.getXML(queryCtx); -- get the next batch
      printClobOut(result);             -- print the next batch of 10 rows..!
    end loop;
  exception
    when others then
    -- dbms_output.put_line(sqlerrm);
       null; -- termination condition, nothing to do;
  end;
  DBMS_XMLQuery.closeContext(queryCtx);  -- close the handle..!
end;
/
```

Setting Stylesheets in XSU (PL/SQL)

The XSU PL/SQL API provides the ability to set stylesheets on the generated XML documents as follows:

- Set the stylesheet header in the result XML. To do this, use `setStylesheetHeader()` procedure, to set the stylesheet header in the result. This simply adds the XML processing instruction to include the stylesheet.
- Apply a stylesheet to the result XML document, before generation. This method is a huge performance win since otherwise the XML document has to be generated as a CLOB, sent to the parser again, and then have the stylesheet applied. XSU generates a DOM document, calls the parser, applies the stylesheet and then generates the result. To apply the stylesheet to the resulting XML document, use the `useStyleSheet()` procedure. This uses the stylesheet to generate the result.

Binding Values in XSU (PL/SQL)

The XSU PL/SQL API provides the ability to bind values to the SQL statement. The SQL statement can contain named bind variables. The variables must be prefixed with a colon (:) to declare that they are bind variables. To use the bind variable follow these steps:

1. *Initialize the query context with the query containing the bind variables.* For example, the following statement registers a query to select the rows from the `emp` table with the where clause containing the bind variables :`EMPNO` and :`ENAME`. You will bind the values for employee number and employee name later.

   ```
   queryCtx = DBMS_XMLQuery.getCtx('select * from emp where empno = :EMPNO and
   ename = :ENAME');
   ```

2. *Set the list of bind values.* The `clearBindValues()` clears all the bind variables set. The `setBindValue()` sets a single bind variable with a string value. For example, you will set the `empno` and `ename` values as shown later:

   ```
   DBMS_XMLQuery.clearBindValues(queryCtx);
   DBMS_XMLQuery.setBindValue(queryCtx,'EMPNO',20);
   DBMS_XMLQuery.setBindValue(queryCtx,'ENAME','John');
   ```

3. *Fetch the results.* This will apply the bind values to the statement and then get the result corresponding to the predicate `empno = 20` and `ename = 'John'`.

   ```
   DBMS_XMLQuery.getXMLClob(queryCtx);
   ```

4. *Re-bind values if necessary.* For example to change the `ENAME` alone to `scott` and reexecute the query,

   ```
   DBMS_XMLQuery.setBindValue(queryCtx,'ENAME','Scott');
   ```

The rebinding of ENAME will now use `Scott` instead of `John`.

XSU Generating XML Example 5: Binding Values to the SQL Statement

The following example illustrates the use of bind variables in the SQL statement:

```
declare
  queryCtx DBMS_XMLquery.ctxType;
  result CLOB;
begin

queryCtx := DBMS_XMLQuery.newContext(
       'select * from emp where empno = :EMPNO and ename = :ENAME');

--No longer needed:
--DBMS_XMLQuery.clearBindValues(queryCtx);
DBMS_XMLQuery.setBindValue(queryCtx,'EMPNO',7566);
DBMS_XMLQuery.setBindValue(queryCtx,'ENAME','JONES');

result := DBMS_XMLQuery.getXML(queryCtx);

--printClobOut(result);

DBMS_XMLQuery.setBindValue(queryCtx,'ENAME','Scott');

result := DBMS_XMLQuery.getXML(queryCtx);

--printClobOut(result);
end;
/
```

Storing XML in the Database Using DBMS_XMLSave

To use DBMS_XMLSave() and XML SQL Utility storage engine, follow these steps:

1. Create a context handle by calling the DBMS_XMLSave.getCtx function and supplying it the table name to use for the DML operations.
2. *For inserts.* You can set the list of columns to insert into using the `setUpdateColNames` function. The default is to insert values into all the columns.

   For updates. The list of key columns must be supplied. Optionally the list of columns to update may also be supplied. In this case, the tags in the XML

document matching the key column names will be used in the WHERE clause of the update statement and the tags matching the update column list will be used in the SET clause of the update statement.

For deletes. The default is to create a WHERE clause to match all the tag values present in each ROW element of the document supplied. To override this behavior you can set the list of key columns. In this case only those tag values whose tag names match these columns will be used to identify the rows to delete (in effect used in the WHERE clause of the delete statement).

3. Supply an XML document to the `insertXML`, `updateXML`, or `deleteXML` functions to insert, update and delete respectively.
4. You can repeat the last operation any number of times.
5. Close the context.

Use the same examples as for the Java case, `OracleXMLSave` class examples.

Insert Processing Using XSU (PL/SQL API)

To insert a document into a table or view, simply supply the table or the view name and then the XML document. XSU parses the XML document (if a string is given) and then creates an INSERT statement, into which it binds all the values. By default, XSU inserts values into all the columns of the table or view and an absent element is treated as a NULL value.

The following code shows how the document generated from the emp table can be put back into it with relative ease.

XSU Inserting XML Example 6: Inserting Values into All Columns (PL/SQL)

This example creates a procedure, `insProc`, which takes in:

- An XML document as a CLOB
- A table name to put the document into

and then inserts the XML document into the table:

```
create or replace procedure insProc(xmlDoc IN CLOB, tableName IN VARCHAR2) is
   insCtx DBMS_XMLSave.ctxType;
   rows number;
 begin
    insCtx := DBMS_XMLSave.newContext(tableName); -- get the context handle
    rows := DBMS_XMLSave.insertXML(insCtx,xmlDoc); -- this inserts the document
```

```
    DBMS_XMLSave.closeContext(insCtx);            -- this closes the handle
end;
/
```

This procedure can now be called with any XML document and a table name. For example, a call of the form:

```
insProc(xmlDocument, 'scott.emp');
```

generates an INSERT statement of the form:

```
insert into scott.emp (EMPNO, ENAME, JOB, MGR, SAL, DEPTNO) VALUES(?,?,?,?,?,?);
```

and the element tags in the input XML document matching the column names will be matched and their values bound. For the code snippet shown earlier, if you send it the following XML document:

```
 <?xml version='1.0'?>
<ROWSET>
  <ROW num="1">
    <EMPNO>7369</EMPNO>
    <ENAME>Smith</ENAME>
    <JOB>CLERK</JOB>
    <MGR>7902</MGR>
    <HIREDATE>12/17/1980 0:0:0</HIREDATE>
    <SAL>800</SAL>
    <DEPTNO>20</DEPTNO>
  </ROW>
  <!-- additional rows ... -->
</ROWSET>
```

you would have a new row in the emp table containing the values (7369, Smith, CLERK, 7902, 12/17/1980,800,20). Any element absent inside the row element would is considered a null value.

XSU Inserting XML Example 7: Inserting Values into Certain Columns (PL/SQL)

In certain cases, you may not want to insert values into all columns. This might be true when the values that you are getting is not the complete set and you need triggers or default values to be used for the rest of the columns. The code that appears later shows how this can be done.

Assume that you are getting the values only for the employee number, name, and job, and that the salary, manager, department number and hiredate fields are filled in automatically. You create a list of column names that you want the insert to work

on and then pass it to the DBMS_XMLSave procedure. The setting of these values can be done by calling setUpdateColumnName() procedure repeatedly, passing in a column name to update every time. The column name settings can be cleared using clearUpdateColumnNames().

```
create or replace procedure testInsert( xmlDoc IN clob) is
  insCtx DBMS_XMLSave.ctxType;
  doc clob;
  rows number;
begin

   insCtx := DBMS_XMLSave.newContext('scott.emp'); -- get the save context..!

   DBMS_XMLSave.clearUpdateColumnList(insCtx); -- clear the update settings

   -- set the columns to be updated as a list of values..
   DBMS_XMLSave.setUpdateColumn(insCtx,'EMPNO');
   DBMS_XMLSave.setUpdateColumn(insCtx,'ENAME');
   DBMS_XMLSave.setUpdatecolumn(insCtx,'JOB');

   -- Now insert the doc. This will only insert into EMPNO,ENAME and JOB columns
   rows := DBMS_XMLSave.insertXML(insCtx, xmlDoc);
   DBMS_XMLSave.closeContext(insCtx);

end;
/
```

If you call the procedure passing in a CLOB as a document, an INSERT statement of the form:

```
insert into scott.emp (EMPNO, ENAME, JOB) VALUES (?, ?, ?);
```

is generated. Note that in the earlier example, if the inserted document contains values for the other columns (JOB, HIREDATE, and so on), those are ignored.

Also an insert is performed for each ROW element that is present in the input. These inserts are batched by default.

Update Processing Using XSU (PL/SQL API)

Now that you know how to insert values into the table from XML documents, let us see how to update only certain values. If you get an XML document to update the salary of an employee and also the department that she works in:

```
<ROWSET>
```

```
    <ROW num="1">
      <EMPNO>7369</EMPNO>
      <SAL>1800</SAL>
      <DEPTNO>30</DEPTNO>
    </ROW>
    <ROW>
      <EMPNO>2290</EMPNO>
      <SAL>2000</SAL>
      <HIREDATE>12/31/1992</HIREDATE>
    <!-- additional rows ... -->
  </ROWSET>
```

you can call the update processing to update the values. In the case of update, you need to supply XSU with the list of key column names. These form part of the `where` clause in the update statement. In the `emp` table shown earlier, the employee number (`EMPNO`) column forms the key and you use that for updates.

XSU Updating XML Example 8: Updating XML Document Key Columns (PL/SQL)

Consider the PL/SQL procedure:

```
create or replace procedure testUpdate ( xmlDoc IN clob) is
  updCtx DBMS_XMLSave.ctxType;
  rows number;
begin

   updCtx := DBMS_XMLSave.newContext('scott.emp');  -- get the context
   DBMS_XMLSave.clearUpdateColumnList(updCtx); -- clear the update settings..

   DBMS_XMLSave.setKeyColumn(updCtx,'EMPNO'); -- set EMPNO as key column
   rows := DBMS_XMLSave.updateXML(updCtx,xmlDoc);  -- update the table.
   DBMS_XMLSave.closeContext(updCtx);             -- close the context..!

end;
/
```

In this example, when the procedure is executed with a `CLOB` value that contains the document described earlier, two update statements would be generated. For the first `ROW` element, you would generate an `UPDATE` statement to update the `SAL` and `JOB` fields as shown:

```
UPDATE scott.emp SET SAL = 1800 and DEPTNO = 30 WHERE EMPNO = 7369;
```

and for the second ROW element,

```
UPDATE scott.emp SET SAL = 2000 and HIREDATE = 12/31/1992 WHERE EMPNO = 2290;
```

XSU Updating XML Example 9: Specifying a List of Columns to Update (PL/SQL)

You may want to specify the list of columns to update. This would speed up the processing since the same update statement can be used for all the ROW elements. Also you can ignore other tags which occur in the document. Note that when you specify a list of columns to update, an element corresponding to one of the update columns, if absent, will be treated as NULL.

If you know that all the elements to be updated are the same for all the ROW elements in the XML document, then you can use the setUpdateColumnName*()* procedure to set the column name to update.

```
create or replace procedure testUpdate(xmlDoc IN CLOB) is
  updCtx DBMS_XMLSave.ctxType;
  rows number;
begin

   updCtx := DBMS_XMLSave.newContext('scott.emp');
   DBMS_XMLSave.setKeyColumn(updCtx,'EMPNO'); -- set EMPNO as key column

   -- set list of columnst to update.
   DBMS_XMLSave.setUpdateColumn(updCtx,'SAL');
   DBMS_XMLSave.setUpdateColumn(updCtx,'JOB');

   rows := DBMS_XMLSave.updateXML(updCtx,xmlDoc); -- update the XML document..!
   DBMS_XMLSave.closeContext(updCtx);   -- close the handle

end;
/
```

Delete Processing Using XSU (PL/SQL API)

For deletes, you can set the list of key columns. These columns will be put as part of the WHERE clause of the DELETE statement. If the key column names are not supplied, then a new DELETE statement will be created for each ROW element of the XML document where the list of columns in the WHERE clause of the DELETE will match those in the ROW element.

XSU Deleting XML Example 10: Deleting Operations for Each Row (PL/SQL)

Consider the delete example shown here:

```
create or replace procedure testDelete(xmlDoc IN clob) is
  delCtx DBMS_XMLSave.ctxType;
  rows number;
begin

   delCtx  := DBMS_XMLSave.newContext('scott.emp');
   DBMS_XMLSave.setKeyColumn(delCtx,'EMPNO');

   rows := DBMS_XMLSave.deleteXML(delCtx,xmlDoc);
   DBMS_XMLSave.closeContext(delCtx);
end;
/
```

If you use the same XML document shown for the update example, you would end up with two DELETE statements,

```
DELETE FROM scott.emp WHERE empno=7369 and sal=1800 and deptno=30;
DELETE FROM scott.emp WHERE empno=2200 and sal=2000 and hiredate=12/31/1992;
```

The DELETE statements were formed based on the tag names present in each ROW element in the XML document.

XSU Example 11: Deleting by Specifying the Key Values (PL/SQL)

If instead you want the delete to only use the key values as predicates, you can use the setKeyColumn function to set this.

```
create or replace package testDML AS
   saveCtx DBMS_XMLSave.ctxType := null;   -- a single static variable

   procedure insertXML(xmlDoc in clob);
   procedure updateXML(xmlDoc in clob);
   procedure deleteXML(xmlDoc in clob);

 end;
/

create or replace package body testDML AS

  rows number;

  procedure insertXML(xmlDoc in clob) is
  begin
    rows := DBMS_XMLSave.insertXML(saveCtx,xmlDoc);
  end;
```

```
  procedure updateXML(xmlDoc in clob) is
  begin
    rows := DBMS_XMLSave.updateXML(saveCtx,xmlDoc);
  end;

  procedure deleteXML(xmlDoc in clob) is
  begin
    rows := DBMS_XMLSave.deleteXML(saveCtx,xmlDoc);
  end;

begin
  saveCtx := DBMS_XMLSave.newContext('scott.emp'); -- create the context once..!
  DBMS_XMLSave.setKeyColumn(saveCtx, 'EMPNO');      -- set the key column name.
end;
/
```

Here a single `delete` statement of the form,

```
DELETE FROM scott.emp WHERE EMPNO=?
```

will be generated and used for all `ROW` elements in the document.

XSU Deleting XML Example 12: Reusing the Context Handle (PL/SQL)

In all the three cases described earlier, `insert, update,` and `delete`, the same context handle can be used to do more than one operation. That is, you can perform more than one `insert` using the same context provided all of those inserts are going to the same table that was specified when creating the `save` context. The context can also be used to mix `updates, deletes, and inserts`.

For example, the following code shows how one can use the same context and settings to `insert, delete, or update` values depending on the user's input.

The example uses a PL/SQL supplied package static variable to store the context so that the same context can be used for all the function calls.

```
create or replace package testDML AS
   saveCtx DBMS_XMLSave.ctxType := null;   -- a single static variable

   procedure insert(xmlDoc in clob);
   procedure update(xmlDoc in clob);
   procedure delete(xmlDoc in clob);

 end;
/
```

```
create or replace package body testDML AS

  procedure insert(xmlDoc in clob) is
  begin
    DBMS_XMLSave.insertXML(saveCtx, xmlDoc);
  end;

  procedure update(xmlDoc in clob) is
  begin
    DBMS_XMLSave.updateXML(saveCtx, xmlDoc);
  end;

  procedure delete(xmlDoc in clob) is
  begin
    DBMS_XMLSave.deleteXML(saveCtx, xmlDoc);
  end;

  begin
    saveCtx := DBMS_XMLSave.newContext('scott.emp'); -- create the context
once..!
    DBMS_XMLSave.setKeyColumn(saveCtx, 'EMPNO');   -- set the key column name.
  end;
end;
/
```

In the earlier package, you create a context once for the whole package (thus the session) and then reuse the same context for performing inserts, updates and deletes.

> **Note:** The key column EMPNO would be used both for updates and deletes as a way of identifying the row.

Users of this package can now call any of the three routines to update the emp table:

```
testDML.insert(xmlclob);
testDML.delete(xmlclob);
testDML.update(xmlclob);
```

All of these calls would use the same context. This would improve the performance of these operations, particularly if these operations are performed frequently.

XSU Exception Handling in PL/SQL

Here is an XSU PL/SQL exception handling example:

```
declare
  queryCtx DBMS_XMLQuery.ctxType;
  result clob;
  errorNum NUMBER;
  errorMsg VARCHAR2(200);
begin

  queryCtx := DBMS_XMLQuery.newContext('select * from emp where df = dfdf');

  -- set the raise exception to true..
  DBMS_XMLQuery.setRaiseException(queryCtx, true);
  DBMS_XMLQuery.setRaiseNoRowsException(queryCtx, true);

  -- set propagate original exception to true to get the original exception..!
  DBMS_XMLQuery.propagateOriginalException(queryCtx,true);
  result := DBMS_XMLQuery.getXML(queryCtx);

  exception
    when others then
      -- get the original exception
      DBMS_XMLQuery.getExceptionContent(queryCtx,errorNum, errorMsg);
      dbms_output.put_line(' Exception caught ' || TO_CHAR(errorNum)
                  || errorMsg );
end;
/
```

Frequently Asked Questions About XML SQL Utility (XSU) for PL/SQL

Here are FAQs about XSU for PL/SQL:

How Can I Use XMLGEN.insertXML with LOBs?

I am trying to use the `insertXML` procedure from XSU. I have little experience with using LOBS. What is the problem in my script?

I have a table `lob_temp`:

```
SQL> desc lob_temp
Name Null? Type
---------------- -------- ------------------  ----------
CHUNK CLOB
```

```
SQL> set long 100000
SQL> select * from lob_temp;

CHUNK
----------------------------------------  ----------
<DOCID> 91739.1 </DOCID>

<SUBJECT> MTS: ORA-29855, DRG-50704, ORA-12154: on create index using
Intermedia
</SUBJECT>
<TYPE> PROBLEM </TYPE>
<CONTENT_TYPE> TEXT/PLAIN </CONTENT_TYPE>
<STATUS> PUBLISHED </STATUS>
<CREATION_DATE> 14-DEC-1999 </CREATION_DATE>
<LAST_REVISION_DATE> 05-JUN-2000 </LAST_REVISION_DATE>
<LANGUAGE> USAENG </LANGUAGE>
```

I have another table where I need to insert data from `lob_temp`:

```
SQL> desc metalink_doc
Name Null? Type
--------------- -------- ----------------- ----------
DOCID VARCHAR2(10)
SUBJECT VARCHAR2(100)
TYPE VARCHAR2(20)
CONTENT_TYPE VARCHAR2(20)
STATUS VARCHAR2(20)
CREATION_DATE DATE
LAST_REVISION_DATE DATE
LANGUAGE VARCHAR2(10)
```

This is the script. It is supposed to read data from `lob_temp` and then insert the data, extracted from the XML document, to table `metalink_doc`:

```
declare
xmlstr clob := null;
amount integer := 255;
position integer := 1;
charstring varchar2(255);
finalstr varchar2(4000) := null;
ignore_case constant number := 0;
default_date_format constant varchar2(21) := 'DD-MON-YYYY';
default_rowtag constant varchar2(10) := 'MDOC_DATA';
len integer;
```

```
insrow integer;
begin
select chunk into xmlstr from lob_temp;
dbms_lob.open(xmlstr,dbms_lob.lob_readonly);
len := dbms_lob.getlength(xmlstr);
while position < len loop
dbms_lob.read(xmlstr,amount,position,charstring);
if finalstr is not null then
finalstr := finalstr||charstring;
else
finalstr := charstring;
end if;
position := position + amount;
end loop;
insrow := xmlgen.insertXML('metalink_doc',finalstr);
dbms_output.put_line(insrow);
dbms_lob.close(xmlstr);
exception
when others then
dbms_lob.close(xmlstr);
dbms_lob.freetemporary(xmlstr);
end;
/
```

This is the error received:

```
ERROR at line 1:
ORA-22275: invalid LOB locator specified
ORA-06512: at "SYS.DBMS_LOB", line 485
ORA-06512: at line 31
ORA-29532: Java call terminated by uncaught Java exception:
oracle.xml.sql.OracleXMLSQLException: Expected 'EOF'.
```

The user I am logged in as owns both tables, and all objects created when I ran `oraclexmlsqlload.csh`.

Answer: You need to have <ROWSET> and <ROW> tags to insert XML document into a table. I modified your procedure. There is a problem when parsing the DATE format, hence I used VARCHAR2:

```
drop table lob_temp;
 create table lob_temp (chunk clob);
 insert into lob_temp values ('
 <ROWSET>
 <ROW>
 <DOCID> 91739.1 </DOCID>
```

```
 <SUBJECT> MTS: ORA-29855, DRG-50704, ORA-12154: on create index using
Intermedia </SUBJECT>
 <TYPE> PROBLEM </TYPE>
 <CONTENT_TYPE> TEXT/PLAIN </CONTENT_TYPE>
 <STATUS> PUBLISHED </STATUS>
 <CREATION_DATE> 14-DEC-1999 </CREATION_DATE>
 <LAST_REVISION_DATE> 05-JUN-2000 </LAST_REVISION_DATE>
 <LANGUAGE> USAENG </LANGUAGE>
 </ROW>
 </ROWSET>
 ');

 drop table metalink_doc;
 create table metalink_doc (
 DOCID VARCHAR2(10),
 SUBJECT VARCHAR2(100),
 TYPE VARCHAR2(20),
 CONTENT_TYPE VARCHAR2(20),
 STATUS VARCHAR2(20),
 CREATION_DATE VARCHAR2(50),
 LAST_REVISION_DATE varchar2(50),
 LANGUAGE VARCHAR2(10)
 );

 create or replace procedure prtest as
 xmlstr clob := null;
 amount integer := 255;
 position integer := 1;
 charstring varchar2(255);
 finalstr varchar2(4000) := null;
 ignore_case constant number := 0;
 default_date_format constant varchar2(21) := 'DD-MON-YYYY';
 default_rowtag constant varchar2(10) := 'MDOC_DATA';
 len integer;
 insrow integer;
 begin

 select chunk into xmlstr from lob_temp;
 dbms_lob.open(xmlstr,dbms_lob.lob_readonly);
 len := dbms_lob.getlength(xmlstr);

 while position < len loop
 dbms_lob.read(xmlstr,amount,position,charstring);
 if finalstr is not null then
 finalstr := finalstr||charstring;
```

```
else
finalstr := charstring;
end if;
position := position + amount;
end loop;

insrow := xmlgen.insertXML('metalink_doc',finalstr);
dbms_output.put_line(insrow);

IF DBMS_LOB.ISOPEN(xmlstr) = 1 THEN
dbms_lob.close(xmlstr);
END IF;

exception
when others then
IF DBMS_LOB.ISOPEN(xmlstr)=1 THEN
dbms_lob.close(xmlstr);
END IF;
end;
/
show err
```

Part V

Tools and Frameworks That Support XDK

This part contains the following chapters, appendixes, and the XML glossary:

24

Developing XML Applications with JDeveloper

This chapter contains the following sections:

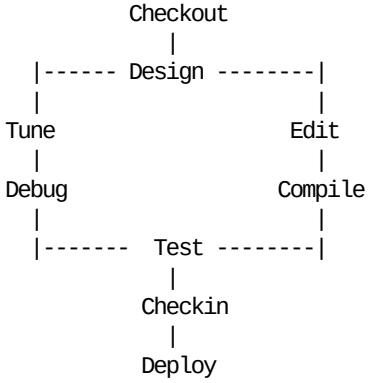
Introducing JDeveloper

Oracle JDeveloper is a J2EE™ development environment with end-to-end support for developing, debugging, and deploying e-business applications. JDeveloper empowers users with highly productive tools, such as the industry's fastest Java debugger, a new profiler, and the innovative CodeCoach tool for code performance analysis and improvement.

To maximize productivity, JDeveloper provides a comprehensive set of integrated tools that support the complete development life cycle, from source code control, modeling, and coding through debugging, testing, profiling, and deploying. JDeveloper simplifies J2EE development by providing wizards, editors, visual design tools, and deployment tools to create high-quality standard J2EE components, including applets, JavaBeans, JavaServer Pages (JSP), servlets, and Enterprise JavaBeans (EJB). JDeveloper also provides a public Addin API to extend and customize the development environment and seamlessly integrate it with external products.

JDeveloper Covers the Complete Development Life Cycle

Java is a relatively new language, and Java development environments are catching up with traditional client/server tools. Developers now require a well-integrated development environment that supports the complete development life cycle:



In a typical scenario, a developer launches JDeveloper, checks out an application from the source control system and starts the development process. UML modelers help the developer with the design of the application, and possibly with the generation of source code. JDeveloper provides wizards and editor, both visual and code-based, to add functionality, and it includes various tools to compile, test, debug, and tune the application. When satisfied, the developer can check the

application back into the source control system and deploy it to the final destination.

JDeveloper Runs on Windows, Linux, and Solaris™ Operating Environment

The 9*i* release of JDeveloper was completely rewritten in Java and now JDeveloper runs on any operating system that has a Java Virtual Machine (JDK 1.3 and later) and will be supported on Windows (NT, 2000, and XP), Linux and Solaris™ Operating Environment.

Another advantage is that the development environment is now fully extensible through the Addin API, which allows customers and third-party vendors to extend the product and integrate it with other products.

Java Alone Is Not Enough

Over the last few years, Java has become *the* programming language for the Internet. Some of the reasons for this popularity are its operating system independence, its simplicity, and its powerful component model.

To build complete e-business applications, however, developers will need more than Java alone. Oracle believes in, and has invested heavily, in the combination of Java, SQL, and XML. Java is used for programming the business and presentation logic, SQL for interacting with the database, and XML for passing information between loosely coupled applications.

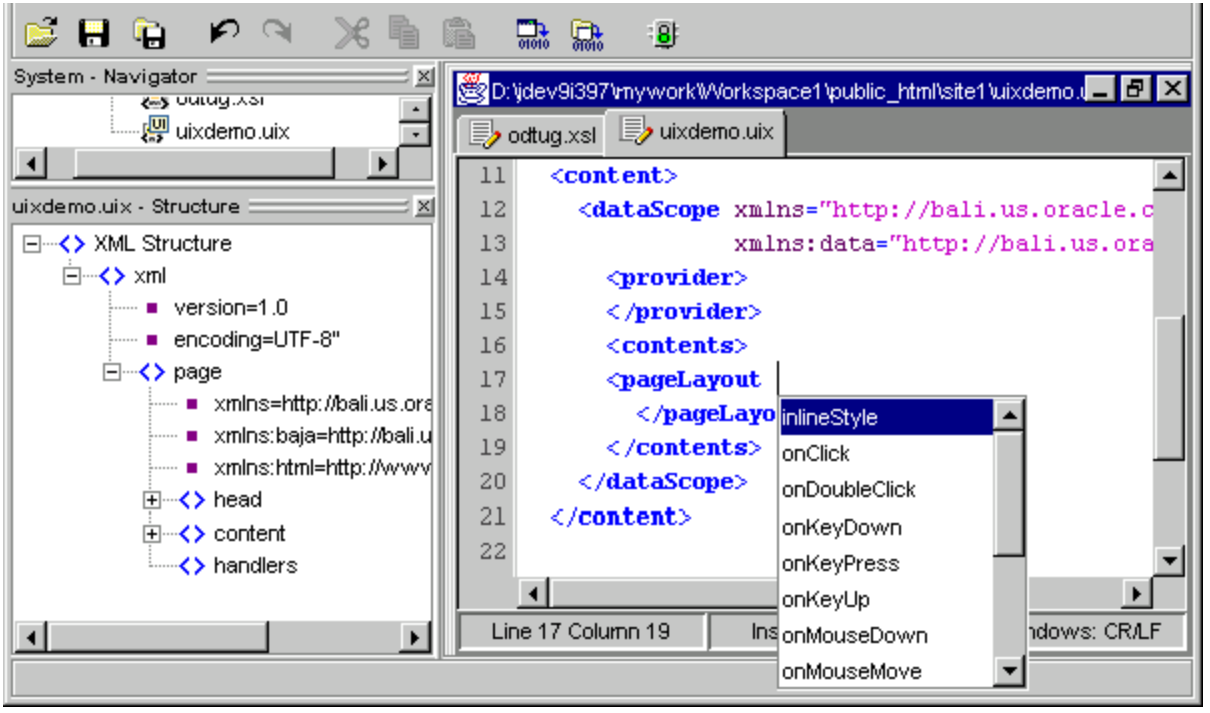
JDeveloper helps developers build e-business applications using Java, XML, HTML, SQL, and PL/SQL. It provides various code editors and visual tools for each of these languages.

XML Tools in JDeveloper

The Oracle XDK is integrated into JDeveloper, offering many ways to create, handle, and transform XML. For example, with the XSQL Servlet, developers can query and manipulate database information, generate XML documents, transform the documents using XSLT stylesheets, and make them available on the Web.

JDeveloper has a new schema-driven XML editor. See Figure 24–1.

Figure 24–1 JDeveloper's Schema-Driven XML Editor in Action



An XML Schema Definition defines the structure of an XML document and is used in the editor to validate the XML and help developers when typing. This feature is called *Code Insight* and provides a list of valid alternatives for XML elements or attributes in the document. Just by specifying the schema for a certain language, the editor can assist you in creating a document in that markup language.

Oracle JDeveloper simplifies the task of working with Java application code and XML data and documents at the same time. It features drag-and-drop XML development modules. These include the following:

- Color-coded syntax highlighting for XML
- Built-in syntax checking for XML and Extensible Style Sheet Language (XSL)
- XSQL Pages and Servlet support, where developers can edit and debug Oracle XSQL Pages, Java programs that can query the database and return formatted XML, or insert XML into the database without writing code. The integrated servlet engine enables you to view XML output generated by Java code in the

same environment as your program source, making it easy to do rapid, iterative development and testing.

- Includes Oracle's XML Parser for Java
- Includes XSLT Processor
- Related XDK for JavaBeans components
- XSQL Page Wizard. See "Page Selector Wizard" on page 24-8.
- XSQL Action Handlers
- Schema-driven XML editor.

Oracle XML Developer's Kit (XDK) is integrated into JDeveloper, so that it offers many utilities to help Java developers handle, create, and transform XML. For example, when designing with XSQL Servlet, you can query and manipulate database information, generate XML documents, transform them using XSLT stylesheets, and make them available on the web.

See Also:

- Chapter 9, "XSQL Pages Publishing Framework"
- `http://jdeveloper.us.oracle.com`
- `http://otn.oracle.com/products/jdev/`
- The online discussion forum for JDeveloper is located at `http://www.oracle.com/forums`

Business Components for Java (BC4J)

To take J2EE application development to a higher level of productivity, JDeveloper now offers Business Components for Java (BC4J), a standards-based, server-side framework for creating scalable, high-performance Internet applications. The framework provides design-time facilities and runtime services to drastically simplify the task of building and reusing business logic.

Oracle Business Components for Java (BC4J) is a 100%-Java, XML-powered framework that enables productive development, portable deployment, and flexible customization of multitier, database-savvy applications from reusable business components.

Application developers use the Oracle Business Components framework and Oracle JDeveloper 's integrated design-time wizards, component editors, and productive

Java coding environment to assemble and test application services from reusable business components.

These application services can then be deployed as either CORBA Server Objects or EJB Session Beans on enterprise-scale server platforms supporting Java technology.

The same server-side business component can be deployed without modification as either a JavaServer Pages / Servlet application or Enterprise JavaBeans component. This deployment flexibility, enables developers to reuse the same business logic and data models to deliver applications to a variety of clients, browsers, and wireless Internet devices without having to rewrite code.

In JDeveloper, you can customize the functionality of existing Business Components by using the new visual wizards to modify your XML metadata descriptions.

See Also: *Chapter 25, "Introduction to BC4J"*

Integrated Web Services Development

JDeveloper integrates standard J2EE development techniques seamlessly with both the latest XML and emerging Web Services Standards (including SOAP, UDDI, and WSDL) and their Java-based equivalents. To preserve existing investments in PL / SQL and J2EE applications, JDeveloper makes it very easy for developers to create, deploy and consume Web Services from J2EE and PL / SQL applications using:

- Web Services creation from Java classes, Enterprise JavaBeans, and PL / SQL procedures.
- Automated WSDL file and SOAP deployment descriptor generation during Web Services creation.
- One-click SOAP service registration and de-registration.
- Support for Oracle9*i* SOAP and Apache SOAP 2.x SOAP servers.
- Web Service proxy creation from WSDL files.
- One-click synchronization of Web Service proxies from WSDL files.
- Server skeleton creation from WSDL files.

What's Needed to Run JDeveloper

JDeveloper is an IDE that has been written in Java and therefore, runs on Windows NT, Windows 2000, Linux and Solaris™ Operating Environment systems. It needs a minimum of 128 Mb RAM.

Minimum system requirements for JDeveloper

Refer to JDeveloper Release Notes. As more products are run on the same machine, system requirements are increased. A typical development environment for running JDeveloper includes:

- Running JDeveloper
- Running Oracle9*i* locally
- Running Oracle9i Application Server locally
- Additional third party tools (profilers, version control, modelers,...)

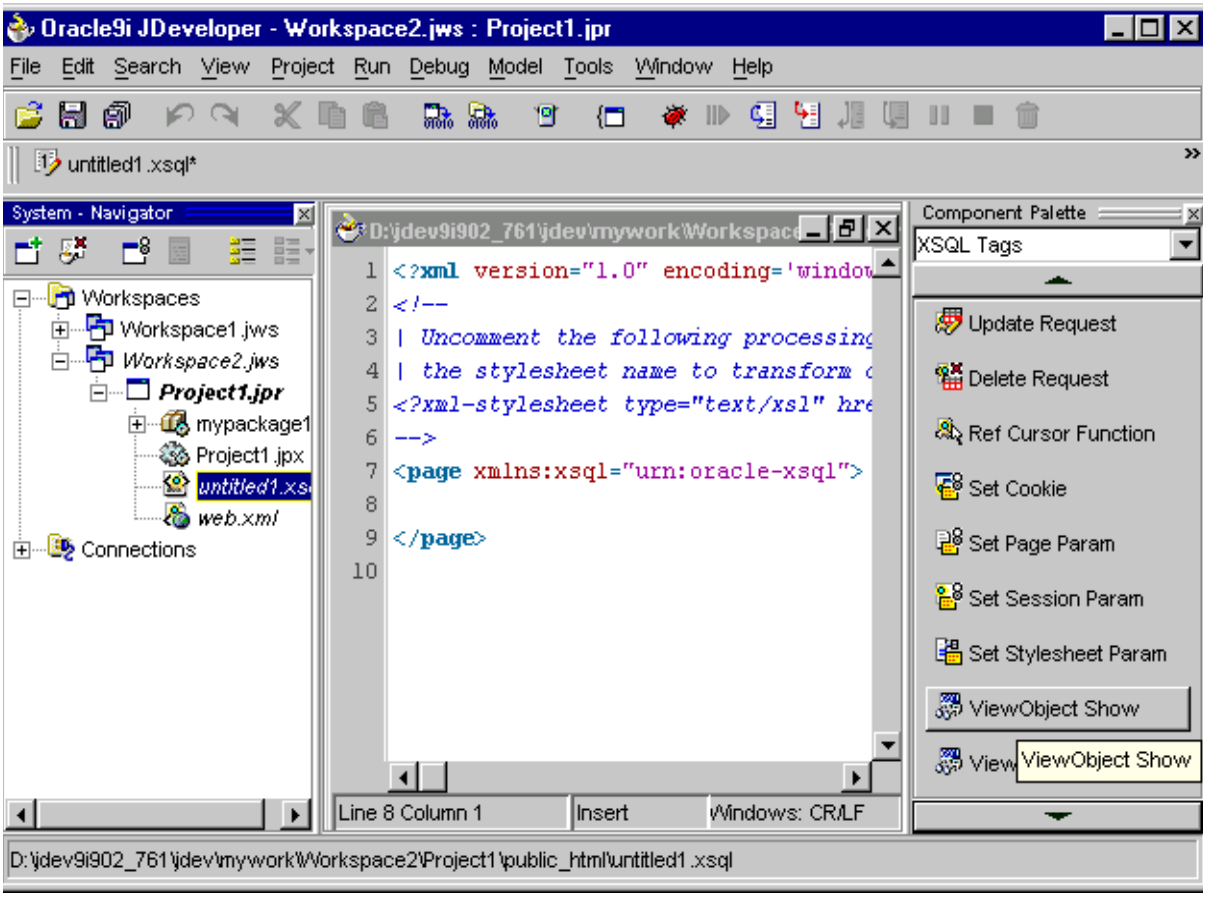
These add to system requirements, in terms of actual CPU usage and in disk space needs.

Business rules can be changed on site without needing access to the underlying component source code.

XSQL Component Palette

XSQL Component Palette provides you with a mechanism to add tags which allows accessing database tables or BC4J View Objects. You can either perform queries against them or update the underlying database tables through them. Figure 24–2, "JDeveloper's XSQL Component Palette" illustrates the JDeveloper XSQL Component Palette.

Figure 24–2 JDeveloper's XSQL Component Palette



Page Selector Wizard

When you need to create XSQL pages while building a web application, you can invoke Page Wizard which enables you to create XSQL Pages on top of either database tables directly or on top of BC4J View Objects. When you choose to build an XSQL Page on top of a BC4J View Object, you are prompted to select an application module from a list or create a new application module and then build the XSQL Pages based application.

See Also: *Oracle9i Java Developer's Guide*

XDK Features in JDeveloper

The following lists JDeveloper's supported XDK for Java components:

- Oracle XML Parser for Java
- Oracle XSQL Servlet

You can use the XML Parser for Java including the XSLT Processor and the XML SQL Utility in JDeveloper as all these tools are written in Java. JDeveloper provides these components.

Sample programs which demonstrate how to use these tools can be found in the `[JDeveloper]/Samples/xmlsamples` directory.

Oracle XDK Integration in JDeveloper

Oracle XDK for Java consists of the following XML tools:

- XML Parser for Java
- XML SQL Utility for Java
- XML Java Class Generator
- XSQL Servlet
- XML Transviewer Beans

All these utilities are written in Java and hence can easily be dropped into JDeveloper and used with no further effort. You can also update the XDK for Java components with the latest versions downloaded from Oracle Technology Network (OTN) at http://technet.oracle.com/tech/xml.

Oracle XDK for Java also includes the XML Transviewer Beans. These are a set of JavaBeans that permit the easy addition of graphical or visual interfaces to XML applications. Bean encapsulation includes documentation and descriptors that make them accessible directly from JDeveloper.

> **See Also:** Chapter 10, "XDK JavaBeans" for more information on how to use the Transviewer Beans.

Developing Web Applications in JDeveloper Using XSQL Pages

The XSQL Servlet is a tool that processes SQL queries and outputs the result set as XML. This processor is implemented as a Java servlet and takes as its input an XML

file containing embedded SQL queries. It uses the XML Parser for Java and the XML SQL Utility to perform many of its operations.

The XSQL Servlet offers a productive and easy way to get XML in and out of the database. Using simple scripts you can:

- Generate simple and complex XML documents
- Apply XSL Stylesheets to generate into any text format
- Parse XML documents and store the data in the database
- Create complete dynamic web applications without programming a single line of code

JDeveloper XSQL Example 1: emp.xsql

For example, consider the following XML example:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="emp.xsl"?>
<FAQ xmlns:xsql="urn:oracle-xsql" connection = "scott">
  <xsql:query doc-element="EMPLOYEES" row-element="EMP">
    select e.ename, e.sal, d.dname as department
    from dept d, emp e
    where d.deptno = e.deptno
  </xsql:query>
</FAQ>
```

Generates the following:

```
<EMPLOYEES>
  <EMP>
    <ENAME>Scott</ENAME>
    <SAL>1000</SAL>
    <DEPARTMENT>Boston</DEPARTMENT>
  </EMP>
  <EMP>
  ...
</EMPLOYEES>
```

With JDeveloper you can easily develop and execute XSQL files. The built in Web Server and the user's default Web Browser will be used to display the resulting pages.

Using Action Handlers in XSQL Pages

XSQL Action Handlers are Java classes which can be invoked from XSQL Page applications very easily. Since these are Java classes they can be debugged from JDeveloper just like any other Java application.

If you are building an XSQL Pages application, you can make use of the XSQL Action Handler to extend the set of actions that can be performed to handle more complex jobs. You will need to debug this Action Handler.

Your XSQL Pages should be in the directory specified in the Project Property "HTML Paths" settings for "HTML Source Directory".

To debug your Action Handler carry out these steps:

1. Assume you have created an `.xsql` file which has reference to a custom Action Handler called MyActionHandler.
2. Debug this Action Handler because it is not exactly behaving as you expect.
3. Set breakpoints in your Java source file.
4. Right mouse click on the `.xsql` file and then choose Debug... from the menu.

See Also: *The JDeveloper Guide* under the online HELP menu.

Building XML Applications with JDeveloper

Consider the following example that demonstrates how XML is used to represent data, not present it. It shows the many to one relationship between employees and departments.

JDeveloper XDK Example 1: BC4J Metadata

```
<Departments>
<Dept>
   <Deptno>10</Deptno>
   <Dname>Sales></Dname>
   <Loc>
   <Employees>
     <Employee>
        <Empno>1001></Empno>
        <Ename>Scott</Ename>
        <Salary>80000</Salary>
     </Employee>
   </Employees>
```

```
...
     </Employee>
   </Employees>
</Dept>
<Dept>
...
```

Procedure for Building Applications in JDeveloper

To build an XSQL project in JDeveloper carry out the following steps:

1. Start a New JDeveloper Project by selecting `File` >`New Project`.
2. Create a Business Components for Java application.
3. Choose `File`>`New` from the menu. Click OK.
4. Choose `WebObjects`>`XSQL` from the menus.
5. Position the cursor between the `<PAGE>` and `<?PAGE>` tags.
6. From Component Palette, choose `ViewObjects Show` tag.
7. Select the application module from the list that pops up.

When you finish these steps in the Page Wizard, you should have an XSQL Page based on the Business Components for Java (BC4J) framework View objects. When you run this page, it sends the XML data to your browser. You could optionally create a stylesheet to format the data so that it appears in a way that you prefer or you can tune it so that it can be displayed on a PDA or cellphone.

Using XSQL Servlet from JDeveloper

XSQL Servlet offers a productive and easy way to get XML in and out of the database.

> **See Also:** Chapter 9, "XSQL Pages Publishing Framework" for information about how to use XSQL Servlet.

When using XSQL Servlet in JDeveloper, you do not need to include the XSQL Runtime in your project as this is already done for any new XSQL Page or XSQL wizard-based application.

Using simple scripts you can do the following from JDeveloper:

- Generate simple and complex XML documents

- Apply XSL stylesheets to generate into any text format
- Parse XML documents and store the data in the database
- Create complete dynamic web applications without programming a single line of code

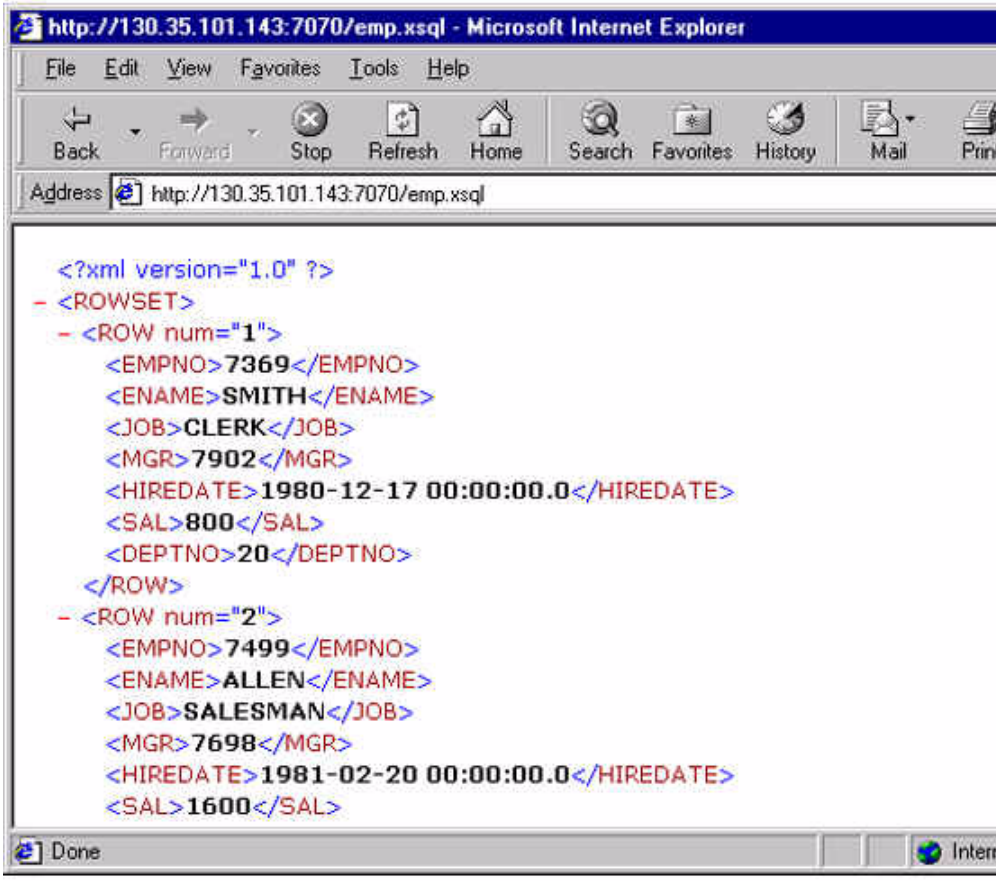
Consider a simple query in an XSQL file, which returns details about all the employees in the emp table. The XSQL code to get this information would be as shown in Example 2.

JDeveloper XSQL Example 2: Employee Data from Table emp: emp.xsql

```
<?xml version="1.0"?>
<xsql:query xmlns:xsql="urn:oracle-xsql" connection="demo">
    select *
    from emp
    order by empno
</xsql:query>
```

Figure 24–3 shows what the raw employee XML data displayed on the browser.

Figure 24–3 Employee Data in Raw XML Format



If you want to output your data in a tabular form, make a small modification to your XSQL code to specify a stylesheet. The changes you would make in this example are shown later highlighted.

JDeveloper XSQL Example 3: Employee Data with Stylesheet Added

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="emp.xsl"?>
<xsql:query xmlns:xsql="urn:oracle-xsql" connection="demo">
    select *
    from emp
```

```
    order by empno
</xsql:query>
```

The result would be a table. You can do a lot more with XSQL Servlet of course.

> **See Also:** Chapter 9, "XSQL Pages Publishing Framework" and also the XDK for Java, XSQL Servlet Release Notes on OTN at http://technet.oracle.com/tech/xml

Frequently Asked Questions About JDeveloper and XML Applications

This section lists JDeveloper questions and answers.

How Do I Construct an XML Document in JSP?

I am dynamically constructing an XML document in a JSP page (from the results of data returned by a PL/SQL API) using the classes generated by the Class generator (based on a DTD) and then applying a XSL stylesheet to render the transformation in HTML. I see that this works fine only for the first time, that is, when the JSP is first accessed (and internally compiled), and fails every time the same page is accessed thereafter.

The error I get is:

```
"oracle.xml.parser.v2.XMLDOMException: Node doesn't belong to the current
document"
```

The only way to make it work again is to compile the JSP, by just 'touching' the JSP page. Of course, this again works only once. I am using Apache JServ.

How can this be overcome? Does the static code in the Java class generated for the top level node have to do anything with it?

Answer: It seems to me that you may have stored some invalid state in your JSP. The XML Parser picks this invalid state, then, throws the exception you mentioned.

As far as I know, CRM does not use an HTTP session in their application. I guess this is true in your case also. You may have used a member variable to store some invalid state unintentionally. Member variables are the variables declared by the following syntax:

```
<%! %>
```

For example:

```
<%! Document doc=null; %>
```

Many JSP users misunderstand that they need to use this syntax to declare variables. In fact, you do not need to do that. In most of cases, you do not need a member variable. Member variables are shared by all requests and are initialized only once in the lifetime of the JSP.

Most users need stack variables or method variables. These are created and initialized for each request. They are declared as a form of scriptlet as shown in the following example:

```
<% Document doc=null; %>
```

In this case, every request has its own `doc` object, and the `doc` object is initialized to `null` for each request.

If you do not store an "invalid" state in session or method variables in your JSP, then there may be other reasons that cause this.

Is There a Way to Use the @code Directly in the document() Line?

Now, if I wish to use the @code as a key, I use

```
<xsl:template match="aTextNode">
    ...
   <xsl:param name="labelCode" select="@code"/>
     <xsl:value-of
 select="document('messages.xml')/messages/msg[@id=$labelCode and
 @lang=$lang]"/>
    ...
   </xsl:template>
```

that works too, but I was wondering if there is a way to use the `@code` directly in the `document()` line?

Answer: This is what the `current()` function is useful for. Rather than:

```
<xsl:param name="labelCode" select="@code"/>
<xsl:value-of
select="document('messages.xml')/messages/msg[@id=$labelCode and
@lang=$lang]"/>
```

you can do:

```
<xsl:value-of
select="document('messages.xml')/messages/msg[@id=current()/@code
```

```
and @lang = $lang]"/>
```

How Do I Retrieve Data from messages.xml?

Is it, or will it be, possible to retrieve the data stored in `messages.xml` from the database? How is the `document()` instruction going to work where listener and servlet will run inside the database?

Answer: Yes. By the spec, the XSLT engine should read and cache the document referred to in the `document()` function. It caches the parsed document based on the string-form of the URI you pass in, so here's how you can achieve a database-based message lookup:

1. `CREATE TABLE MESSAGES (lang VARCHAR2(2), code NUMBER, message VARCHAR2(200));`
2. Create an XSQL page like `msg.xsql`:

```
<xsql:query lang="en" xmlns:xsql="urn:oracle-xsql" connection="demo"
            row-element="" rowset-element="">
      select message
        from messages
       where lang = '{@lang}'
         and code = {@code}
</xsql:query>
```

3. Create a stylesheet that uses `msg.xsql` in the `document()` function as in this example:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
  <xsl:template match="/">
  <html><body>
    In English my name is
      <xsl:call-template name="msg">
         <xsl:with-param name="code">101</xsl:with-param>
      </xsl:call-template><br/>
    En espanol mi nombre es
      <xsl:call-template name="msg">
         <xsl:with-param name="code">101</xsl:with-param>
         <xsl:with-param name="lang">es</xsl:with-param>
      </xsl:call-template><br/>
    En fran&#231;ais, je m'appelle
      <xsl:call-template name="msg">
         <xsl:with-param name="code">101</xsl:with-param>
         <xsl:with-param name="lang">fr</xsl:with-param>
```

```
      </xsl:call-template><br/>
    In italiano, mi chiamo
      <xsl:call-template name="msg">
         <xsl:with-param name="code">101</xsl:with-param>
         <xsl:with-param name="lang">it</xsl:with-param>
      </xsl:call-template>
  </body></html>
  </xsl:template>
  <xsl:template name="msg">
    <xsl:param name="lang">en</xsl:param>
    <xsl:param name="code"/>
    <xsl:variable name="msgurl"
select="concat('http://xml/msg.xsql?lang=',$lang,'&amp;code=',$code)"/>
    <xsl:value-of select="document($msgurl)/MESSAGE"/>
  </xsl:template>
</xsl:stylesheet>
```

4. Try it out at `http://xml/testmessage.xsql`

This is great if you want to fetch the message from over the web. Alternatively, you could use the msg.xsql preceding but include it in your XSQL Page if that makes sense using:

```
<xsql:include-xsql href="msg.xsql?lang={@lang}&amp;code={@code}"/>
```

Or you could write your own custom action handler to use JDBC to fetch the message and include it in the XSQL page yourself.

How Do I Move Complex XML Documents to a Database?

I am moving XML documents to an Oracle database. The documents are fairly complex. Can an XML document and the Oracle Developer's Kit (XDK) generate a possible DDL format for how the XML Document should be stored in the database, ideally generating an Object-Relational Structure. Does anyone know of a tool that can do this?

Answer: The best way may be to use the Class Generator. Use the XML SQL Utility (XSU) if DTD files are not already created. You'll still have to write a mapping program.

Another method is to create views and write stored procedures to update multiple tables. Unfortunately, you'll have to create your tables and views beforehand in either case.

25

Introduction to BC4J

This chapter contains the following sections:

Introducing Business Components for Java (BC4J)

Business Components for Java is JDeveloper's programming framework for building multitier database applications from reusable business components. Such applications typically consist of:

- A client-side user interface written in Java and/or HTML.
- One or more business logic tier components that provide business logic and views of business objects.
- Tables on the database server that store the underlying data.

A multitier application built with the Business Components for Java framework deploys views, business rules, and custom code in components that clients can share. With the Business Components for Java framework, such components are easy to build and maintain, easy to use and reuse, and easy to customize. Components do not need modification to be deployed to any supported platform.

> **See Also:** Figure 25–1, "Using Business Components for Java (BC4J)" for one example of a multitier application.

This approach provides many features and benefits, including:

Table 25–1 Features and Benefits of BC4J

Feature	Description
Encapsulated business logic	Business logic, including validation, resides and executes in the business logic tier, enabling truly thin clients, easy customizing, and reuse.
Flexible views of data	Views of data are SQL-based and completely separate from the underlying entities, enabling flexible presentation schemes.
Thin clients	BC4J supports thin clients--simple windows to business logic and views of data processed by the business logic tier.
flexible deployment	Deploy locally or on standard server platforms as CORBA server objects and EJB Session Beans.
Database interaction	BC4J's component-based framework handles many repetitive coding tasks, such as master-detail coordination and locking.
Transaction management	Business Components for Java manages changes in its cache and handles posting of changes to the database.

BC4J comprises a framework for building and customizing domain-specific components. As a developer, you derive objects from the classes and interfaces provided by the framework and add custom code to implement features specific to your application. The following business components are used to support this process:

Table 25–2 Business Components in BC4J

Object	Description
Entity object	An entity object encapsulates business logic for a database table, view or synonym. Clients access an entity object's data through one or more view objects. A given entity object can be used by any number of view objects. Relationships between entity objects are expressed using associations.
View object	View objects use SQL queries to specify filtered subsets of attributes from entity objects. Clients manipulate data by navigating through the result set, getting and setting attribute values. Relationships between view objects are expressed using view links.
Application module	An application module is a logical container for instances of view objects, view links, and transactions specified by other application modules.

Each business component you create is represented by an XML file and one or more Java files. The XML file stores metadata (the descriptive information about features and settings of an application you declare using wizards at design time), while the Java file stores the object's code (which implements application-specific behavior). Each object is organized into a package using the directory-based semantics of packages in Java.

The Java and XML files that represent business components use a similar syntax to identify the package they are part of:

Table 25–3 Java and XML Syntax Used by BC4J

Java	XML

package d2ePackage;
...
public class DeptViewImpl extends
oracle.jbo.server.ViewObjectImpl {
...
}
	<ViewObject
	Name="DeptView"
	...
	ComponentClass="d2ePackage.DeptViewImpl">

What Is the Business Components Framework?

The business components framework is a class library, in `oracle.jbo.*`, with built-in application functionality. Using the framework involves specializing base classes to introduce application-specific behavior, allowing the framework to coordinate many of the basic interactions between objects.

By using the Business Components for Java design-time wizards and editors, you can build business logic tiers by defining the characteristics of components: their attributes, relationships, and business rules. Business Components for Java generates Java source code and XML metadata to implement the behavior you have specified. Because the code inherits from a framework, the Java source files are concise and do not contain large amounts of generated code, so it's easy to see where to add the code that models your business. You can use JDeveloper to add the Java code to enhance or change the behavior, and easily test the application services, independently of the deployment platform.

Using Business Components

JDeveloper provides integrated support for the Business Components for Java framework. Using design tools such as wizards and property editors you define the characteristics of objects: their attributes, relationships, and business rules. Then JDeveloper generates executable Java code and XML to implement the behavior you define for the components.

In theory, you could write this code yourself. In practice, though, it's better to use the wizards to be sure that all necessary code is generated and all dependencies are addressed. Then you can edit the generated code to meet the specific needs of your applications. JDeveloper enforces no particular methodology, but the development process typically involves answering questions like these:

- What are the entities and business objects? You can use entity objects on their own (for example, a customer), or you can combine several entity objects (for example, a purchase order consisting of a header, line items, shipments, and distributions).
- How are the entities related? For example, you could define a one-to-many association between departments and employees.
- What are the validation rules? For example, a business rule might specify a minimum salary for employees with more than five years of service. You can apply rules to attributes, entities, and business objects.
- What data will be presented and manipulated? By creating views, you define SQL queries to select and filter data from the entities to minimize network traffic and client-side processing requirements.

Advantages at BC4J Design Time

1. Real-world entities (for example, employees) are used to represent data stored in tables in a database.

2. JDeveloper uses data and metadata from the table to create a Java class that represents the entity. You can edit this Java code to change the default attributes and behavior.

3. JDeveloper also represents metadata in a customizable XML file.

4. JDeveloper can create default view objects to specify criteria for selecting data. You can define your own view objects in addition to (or instead of) the defaults.

5. JDeveloper generates customizable Java classes for each view object: a class for the view object definition and a class for the row. It also generates an XML file for each view object.

6. You use a wizard to define an application module. An application module is a logical container for related objects. It provides a context for defining and executing transactions.

After the application service comprising the business components is designed, built, tested, and debugged, you can deploy it.

Advantages at BC4J Runtime

1. Client code initializes an application module, loading the entities and views it contains.

2. When a view object executes a query at run time, it manipulates data from the corresponding entity or entities.

3. Each view object provides a default iterator that you can use to navigate through its result set.

4. When a query fetches one or more result rows, individual rows are represented by Row objects. Each column value in the result row is accessed through an attribute of a Row object.

5. Controls in the client form enable users to view and edit the data. The controls display rows provided by view objects, which are themselves bound to underlying entity objects. So, when a user changes a value in a control, the Business Components for Java framework sends the action to the view object, which sends it to the entity object. Business rules (if any) attached to the entity object validate the new value before the framework sends it to the database.

Implementing XML Messaging

The Business Components for Java (BC4J) framework provides a general, metadata-driven solution for mapping E-commerce XML Messages into and out of the database.

Sun Microsystems, Inc. provides a Java Message Service (JMS) API, and Oracle9i provides an Advanced Queueing API, that you can use with Business Components for Java to implement XML messaging.

To do so, you use business component `framework methods in the ViewObjectImpl` and ViewRowImpl classes which enable the reading and writing of a canonical format of XML data:

- `writeXML()` - Writes the current object into an XML Element, which can be added to any XML Document, including as a payload for an XML Message.
- `createXMLDefinition()`- Creates an XML DTD for a ViewObject or ViewRow.
- `readXML()` - Reads the attribute values or rows in this object from the XML Element, which could be derived from an XML Document or an XML Message.

The XML messaging sample shows you how to implement a working messaging system. In addition, it provides the general steps you need to follow to implement XML messaging. See `$ORACLE_HOME\BC4J\samples`.

For more information on business component methods, see the Javadoc. For more information on JMS, see the Javasoft web site. For more information on Advanced Queueing, see your Oracle9i documentation.

Test BC4J Applications using JDeveloper

You can use Oracle BC4J framework and Oracle JDeveloper 's wizards and component editors to assemble and test application services from your reusable business components.

In JDeveloper, you can also customize the functionality of existing Business Components by using the visual wizards to modify your XML metadata descriptions.

See Also:

- Chapter 21, "XSLT Processor for PL/SQL"
- *Oracle9i Java Developer's Guide*
- http://otn.oracle.com/products/bc4j

BC4J Uses XML to Store Metadata

The business components for Java framework that ships with JDeveloper uses XML to store metadata about its application components. Important information is now stored in a structured document rather than in Java source code. This makes the application easier to understand and customize.

The application is now customizable without having access to the source code.

Figure 25–1, "Using Business Components for Java (BC4J)" shows how you use BC4J to generate XML documents.

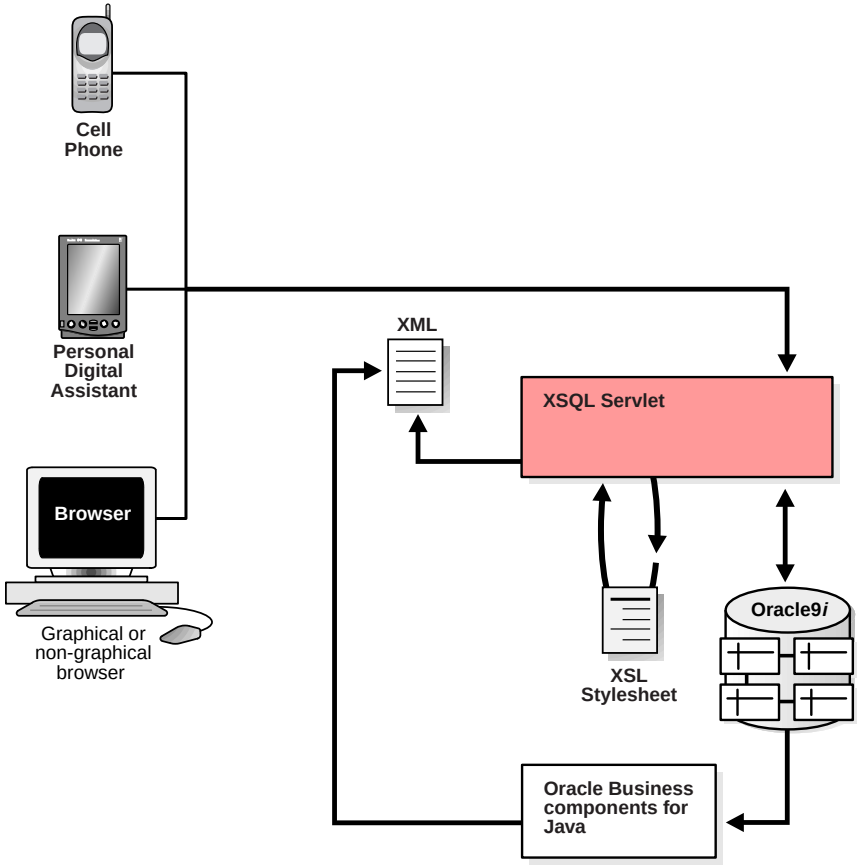
BC4J framework provides a general, metadata-driven solution for mapping e-commerce XML messages into and out of the database. BC4J has a technical white paper on its features available at the following Web site:

```
http://otn.oracle.com/products/jdev/content.html.
```

BC4J is a pure-Java, XML-based business components framework for making building e-commerce applications easier. It is a Java framework usable on its own, but also has tight development support built-into JDeveloper, available for download from the same Web site:

BC4J lets you flexibly map hierarchies of SQL-based view components to underlying business components that manage all application behavior (rules and processes) in a uniform way. It also supports dynamic functionality, so most of its features can be driven completely off XML metadata. You can build a layer which flexibly maps any XML document into and out of the database using this framework. One key benefit is that when XML Documents are put into the system, they automatically can have all the same business rules validated.

Figure 25–1 Using Business Components for Java (BC4J)



Business rules can be changed on site without needing access to the underlying component source code.

Creating a Mobile Application in JDeveloper

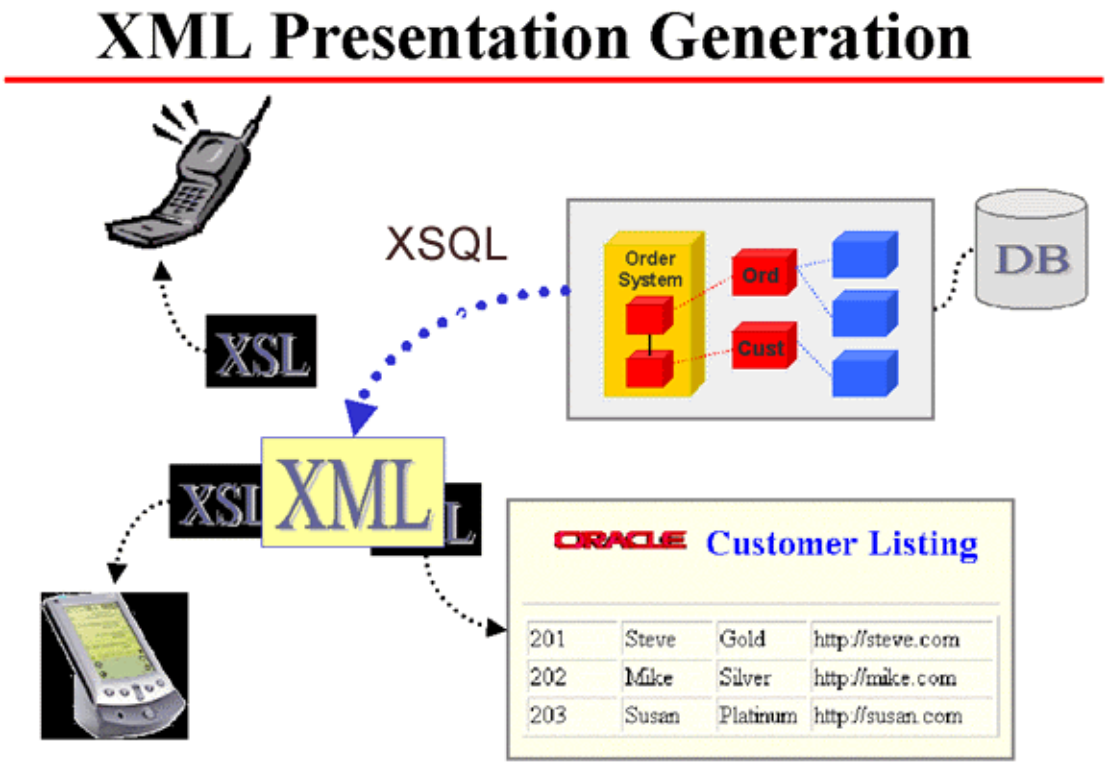
This mobile application is a Departments database application that demonstrates how Business Components for Java (BC4J) and XML can be used to develop applications that can be accessed over wireless devices. The application consists of two main parts:

- Server-side business logic which is developed using the Business Components for Java (BC4J) Framework and the second is the client part. The business logic consists of a view object based on the DEPT table in SCOTT's schema.
- A mechanism to query the DEPT table and update it from any client device including a browser, a cellular phone and a Palm Pilot. For the latter device, the application uses emulators running on Windows NT.

Figure 25–2, "Creating a Mobile Application in JDeveloper Using BC4J and XSQL Servlet" shows schematically how the mobile application works with BC4J, XSQL Servlet, XSL Stylesheets, and Oracle9*i*.

You can see a more comprehensive demo of a similar application on `http://otn.oracle.com/tech/xml`.

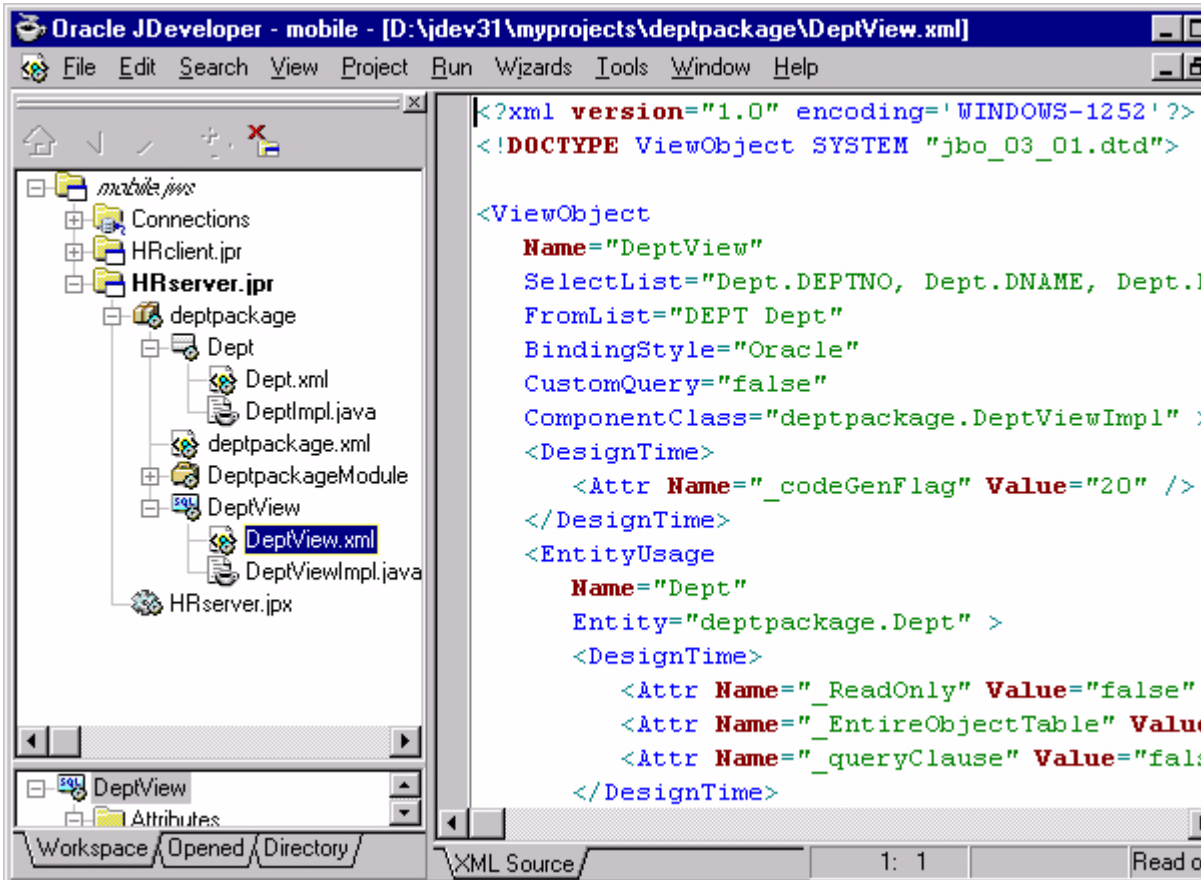
Figure 25–2 Creating a Mobile Application in JDeveloper Using BC4J and XSQL Servlet



Create the BC4J Application

First create the BC4J application. It connects to the SCOTT schema on an Oracle9*i* database. Figure 25–3, "BC4J Application: DEPT View Object XML File" shows the XML file containing the metadata about the DEPT object. See "JDeveloper XDK Example 1: BC4J Metadata" on page 24-11.

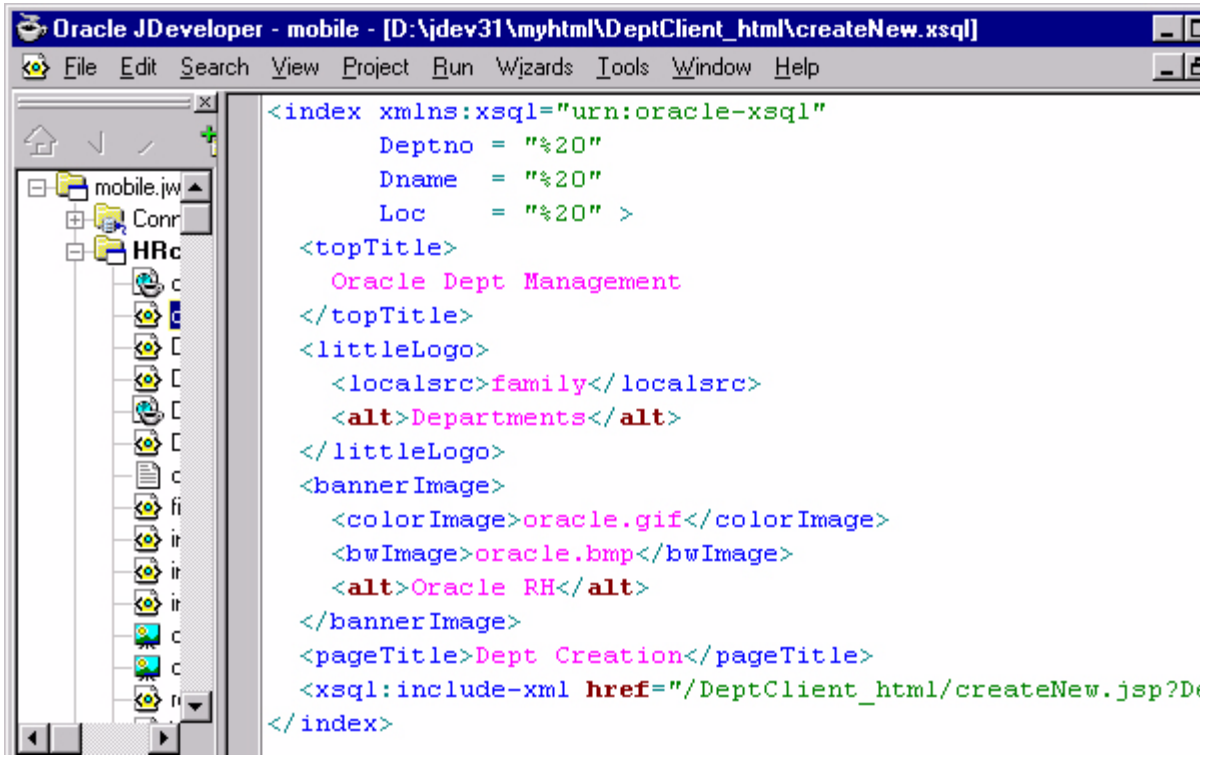
Figure 25–3 BC4J Application: DEPT View Object XML File



Create JSP Pages Based on a BC4J Application

You can then create JSP pages based upon this BC4J application. In the JSP pages you are introduced to the XML Data Generator Web Beans. Figure 25–4, "BC4J Application: XSQL File Calling JSP Page" shows the XSQL file which calls the JSP page to create the new department.

Figure 25–4 BC4J Application: XSQL File Calling JSP Page



Create XSLT Stylesheets According to the Devices Needed to Read the Data

We create XSLT stylesheets to go with the various client devices that we are going to access our data from. In your XSQL files, you specify the list of stylesheets and the protocols they go with which basically ties the stylesheets to the client device.

Example 25–5, "BC4J Application: XSL Stylesheet (indexPP.xsl)" shows an example code snippet of a stylesheet (`indexPP.xsl`) which transforms the XML data to HTML for displaying on a browser on the Palm Pilot emulator.

Figure 25–5 BC4J Application: XSL Stylesheet (indexPP.xsl)

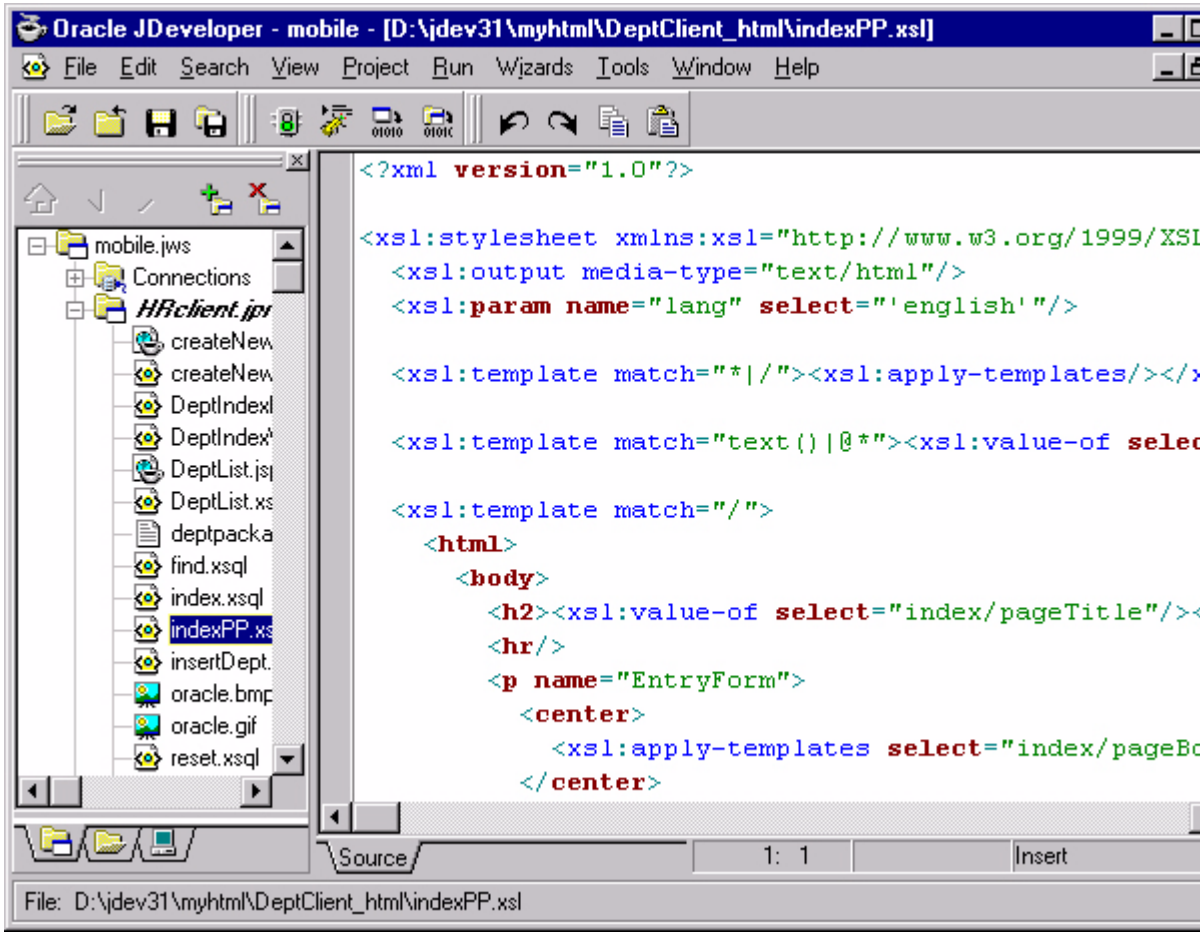


Figure 25–6, "Cell Phone Emulator Running the Department Application Client" shows the Cell Phone Emulator running the Departments Application Client. It also shows the setup screen for the Cell Phone Emulator.

Figure 25–6 Cell Phone Emulator Running the Department Application Client

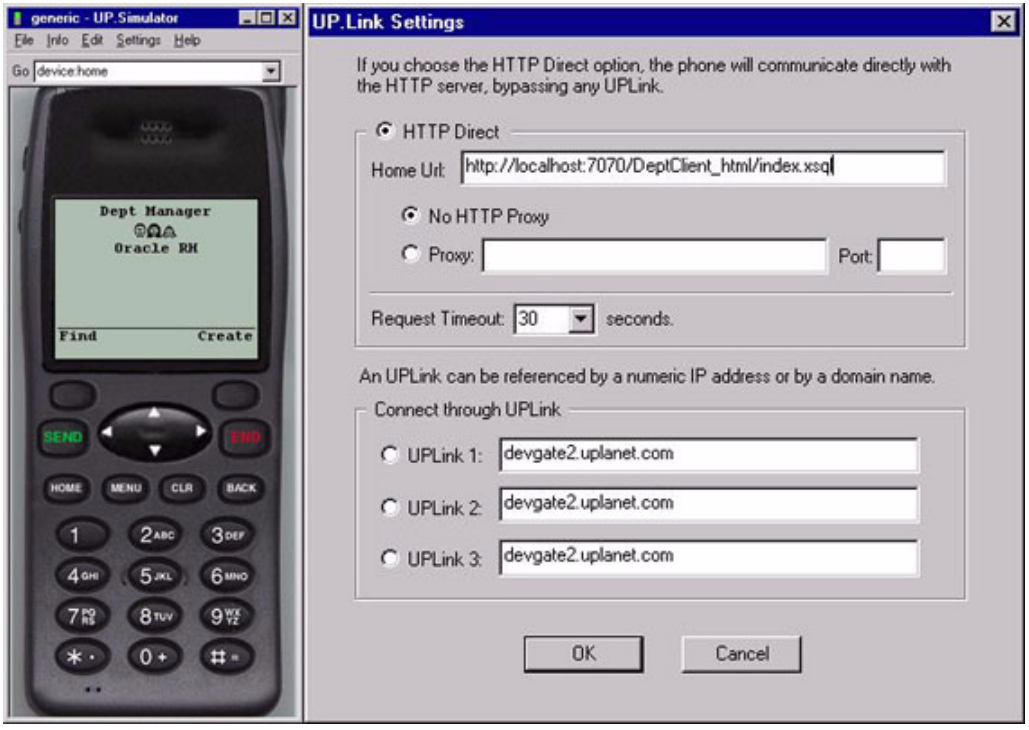


Figure 25–7, "Palm Pilot Emulator Accessing the BC4J Departments Application Through HandWeb Browser" shows the Palm Pilot Emulator accessing the Departments Application by means of HandWeb Browser.

Figure 25–7 Palm Pilot Emulator Accessing the BC4J Departments Application Through HandWeb Browser

Building XSQL Clients with BC4J

In JDeveloper9i, you can build XSQL Pages which can integrate with BC4J application modules and thereby serve application logic from the middle tier to multiple clients. You can retrieve XML data and present it to any kind of a client device just by applying the corresponding stylesheet.

See Also:

- Chapter 21, "XSLT Processor for PL/SQL"
- *Oracle9i Java Developer's Guide*
- *Oracle9i XML Case Studies and Applications*

Building XSQL Clients with BC4J

In JDeveloper 9i, you can build XSQL Pages which can integrate with BC4J application modules and thereby serve application logic from the middle tier to

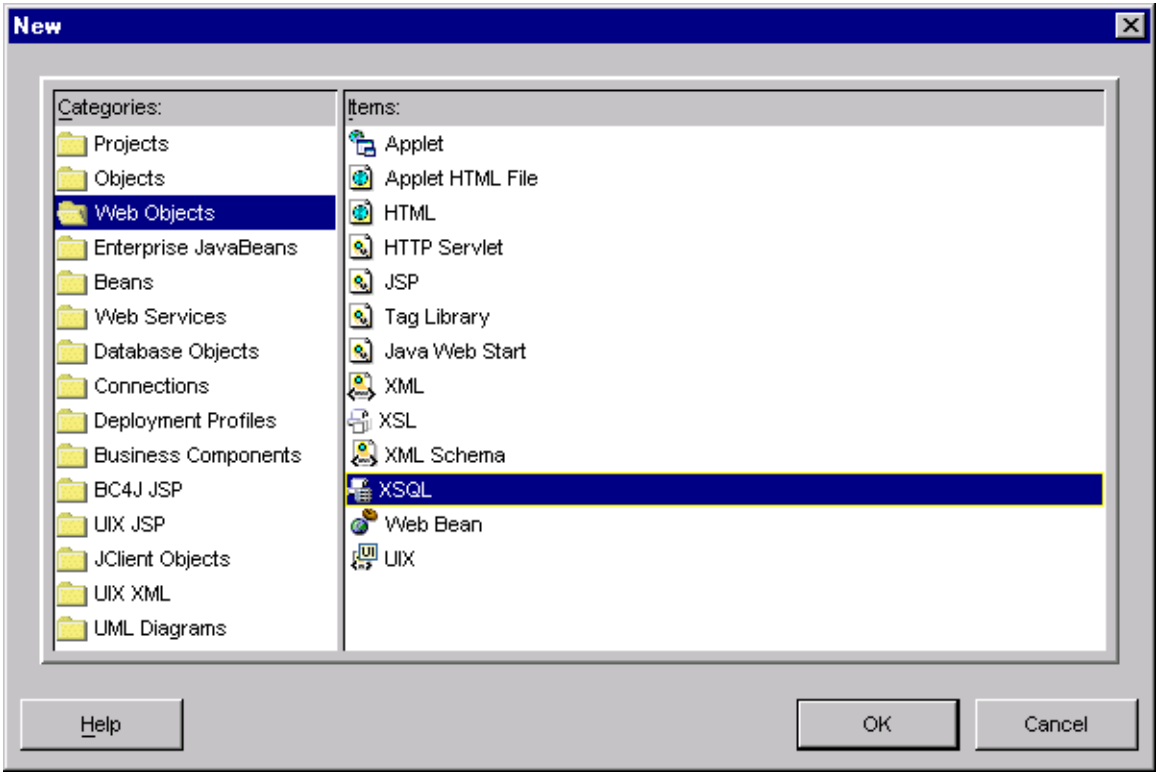
multiple clients. You can retrieve XML data and present it to any kind of a client device just by applying the corresponding stylesheet.

Web Objects Gallery

The Web Objecst Gallery has icons to assist in creating XSQL, XML, and XSL documents easily. When you click them, the basic tags for these pages are generated and you can then enhance them.

The XSQL Pages icon is of special interest because the XSQL Component Palette can be used, after generating your basic XSQL pages, to insert data bound tags in the XSQL pages. Figure 25–8 illustrates JDeveloper's Web Objects Gallery.

Figure 25–8 JDeveloper's Object Gallery Showing the new XSQL, XML, and XSL Icons



Generating and Managing Code When Building XML and Java Applications

The following lists some typical JDeveloper code requirements when using the BC4J framework to build an XML application:

- A `.java` file and a `.xml` file for each entity object and each view object
- A `.java` file for each association object and each link object
- A `.java` file and a .xml file for the application module
- Double-click any of these files in the JDeveloper navigator to view the file contents.

The BC4J framework represents each Business Component that uses a combination of XML and Java code.

- *XML*. The XML code defines the metadata representing declarative settings and features of the object.
- *Java*. The Java code implements the object's behavior.

Other typical generated files are:

- Java implementation of the entity
- View XML file
- Java implementation of the view
- Application module XML file
- Java implementation of the application module

Frequently Asked Questions for BC4J

Some FAQs for BC4J are:

Can Applications Built Using BC4J Work With Any J2EE-Compliant Container?

Answer: Yes. The BC4J framework works with any J2EE-compliant application server. The Oracle9i JDeveloper IDE supports automatically packaging a BC4J-powered J2EE application for deployment to any J2EE 1.2 container. In addition, if you are using Oracle9iAS or WebLogic containers, in addition to this packaging assistance, the tool can automatically carry out the deployment for you, too.

Can J2EE Applications Built Using BC4J Work with Any Database?

Answer: Yes. Any SQL92-compatible database.

By default, the BC4J framework takes specific advantage of the Oracle database and features of the Oracle JDBC Driver to maximize application performance. However, by using the runtime-configurable "SQL Flavor" parameter, applications built with

BC4J can target non-Oracle databases as well. In particular, the Oracle9i JDeveloper release of the BC4J framework has been tested against IBM's DB2 database and Microsoft's SQL Server database (using Merant DataDirect drivers).

Is There Runtime Overhead from the Framework for Features That I Do Not Use?

Answer: No. The BC4J framework has been carefully designed and optimized to avoid runtime overhead for features of the framework that are not being used. For example, BC4J entity objects are designed to encapsulate business logic and handle persistence. If you use them only to handle persistence, perhaps leaving business logic enforcement to existing database triggers in your database, then you do not pay runtime overhead for business logic enforcement that you are not using. Similarly, the BC4J framework supports various kinds of lightweight listeners that developers can use to be notified when interesting framework life cycle events occur. Again, if there are no event subscriptions, there is no overhead associated.

Where Can I Find More Information About BC4J?

Answer: For additional information on BC4J and JDeveloper, please visit:

`http://jdeveloper.us.oracle.com`

For a good technical overview white paper of how BC4J can help J2EE and EJB developers be more productive, please see:

`http://otn.oracle.com/products/jdev/htdocs/j2ee_with_bc4j/j2ee_with_bc4j.html`

26

Introduction to UIX

This chapter contains the following sections:

What Is UIX?

UIX (User Interface XML) is a set of technologies that constitute a framework for building web applications. The main focus of UIX is the user presentation layer of an application, with additional functionality for managing events and for managing the state of the application flow. UIX is designed to create applications with page-based navigation, such as an online human resources application, rather than full-featured applications requiring advanced interaction, such as an integrated development environment (IDE).

An application can interact with UIX in predefined places called decision points, where a decision is made by the operator or a certain action routine is automatically triggered. Execution of an action terminates in a new decision point. The application's structure is provided to UIX in configuration files, which can be ASCII files, databases, or resource files.

The main focus of UIX is the user presentation layer of an application, with additional functionality for managing events and for managing the state of the application flow. UIX is designed to create applications with page-based navigation, such as an online human resources application, rather than full-featured applications requiring advanced interaction, such as an integrated development environment (IDE).

UIX includes Java class libraries, APIs, XML languages, and other technologies for developing different aspects of web-based applications. You can use some or all of these technologies, depending on what aspects of a web application you are developing. It is worthwhile to familiarize yourself with all the UIX technologies to make sure you take full advantage of what they provide.

When to Use UIX

Here are the features of using UIX that make for more rapid development:

- UIX provides an open, flexible framework for development. You can choose among the different UIX technologies for different development needs. For instance, you can use UIX components for rendering pages, or you can use your own HTML or Java Server Pages (JSP) for rendering while still taking advantage of the remaining features of UIX. Additionally, you can use whatever back-end data technologies that best suit your needs.
- The UIX technologies are platform independent because they are implemented in the Java programming language and other portable web technologies.

- UIX supports a wide range of client agents. UIX will adjust its presentation for various browsers and locales. It also supports rendering for mobile devices.
- Applications written to the UIX technology stack maintain a consistent appearance. The UIX rendering projects implement high level user interface controls, which are consistently rendered across your application (and the applications of others using UIX).
- UIX applications are customizable at multiple levels. You can change many aspects of the application independently, including page layout, styles, and imaging. The environment makes simple customizations easy, and more complicated customizations possible.
- If you choose, much of your UIX development can be declarative. This is because the framework can derive its page layouts, styles, and many other features from XML documents, with no programming or compiling involved.
- The UIX architecture has been designed with localization and internationalization support in mind. Its rendering technologies automatically adjust for the target client's locale, and the framework is built to help separate localizable content from the user interface.
- High performance has been designed into the framework, such as the caching and reuse of shared resources.

When Not to Use UIX

These are some cases where it is inappropriate to use UIX:

- If your target user environments have no Java requirements and they can be standard web browsers or mobile devices, using UIX may not be justified. The reason is that the deployment environment for your application must support a Java Virtual Machine (JVM), because UIX is built in Java.
- If your user interface requires advanced interactions such as drag-and-drop, code editing, or visual design, you should use a more complicated user interface technology than UIX provides, such as client-side Java.

What Are the UIX Technologies?

The UIX technologies can be used to implement the entire presentation layer of a web application. However, you can use only a subset of UIX if you only need some of its features. UIX is modularized into "subproducts" that target different aspects of a web application development project. Each is described briefly next.

UIX Components

UIX Components comprise a class library for generating the content of pages, in particular, pages used as the front end (user interface) to a web application. This technology does not manage the navigation between pages or the data supplied to those pages; that functionality is deferred to other sources (such as other UIX technologies). Instead, the UIX Components technology focuses on the rendering of a page itself. This rendering can be HTML for a browser page, or another technology such as WML for a mobile device.

The UIX Components technology does this by including a collection of web beans (or "nodes") for creating page layouts and standard user interface objects, such as tables, tabs, and buttons. It also includes a set of rendering classes (Renderers) that generate output using these Beans for a particular device, such as a browser.

UIX Components have a pluggable rendering architecture that enables rendering the same page with alternative visual styles (that is, the "look and feel"). The default renderers output HTML that conforms to the Oracle Browser Look and Feel (BLAF), but other renderers are available for mobile devices, and additional renderers can be created and added to the framework as needed.

The Java code and classes supporting UIX Components are all located in the `oracle.cabo.ui` package and its subpackages.

UIX Controller

UIX Controller is a framework for developing web application flow. UIX Controller is based on the Java Servlet technology, a standard part of the Java 2 Enterprise Edition. Where UIX Components focus on rendering a given page, UIX Controller is designed to manage the navigation among all pages in an application. UIX Controller defers the rendering of those pages to other technologies (such as UIX Components).

UIX Controller standardizes the way applications deal with HTML events and provides built-in services such as error page loops, login support, and file uploading. While it operates independently of the technology used to render individual pages -- such as UIX Components, JSPs, or Extensible Stylesheet Transformations (XSLT) -- it has built-in support to ease development when technologies like UIX Components are used.

The Java code and classes supporting UIX Controller are all located in the `oracle.cabo.servlet` package and its subpackages.

UIX Language

The UIX language is a declarative alternative to creating web applications programmatically with Java-based UIX Components Beans and/or UIX Controller Java code. The UIX language builds on top of UIX Components and UIX Controller, providing an XML language for specifying UIX Components page layouts and UIX Controller server-side events. Essentially, the UIX language lets you create UIX Components pages and UIX Controller events with an XML document, rather than through Java programming.

While the UIX language provides an alternative way for you to create pages and page flows, it is transformed into UIX Components and UIX Controller objects behind the scenes and is thus treated equally by UIX.

The Java code and classes supporting the UIX language are all located in the `oracle.cabo.ui.xml` and `oracle.cabo.servlet.xml` packages and their subpackages.

UIX Dynamic Images

UIX Dynamic Images describes a utility for generating images that contain text, including built-in support for buttons and tabs. UIX Dynamic Images can colorize the images of an application to support color schemes, as well as provide localization and accessibility support and provide caching support for improved performance. UIX Dynamic Images generate images and, for those who need them, image maps.

Because text is processed separately from images, localization with UIX Dynamic Images is easier and more efficient. Translators work only with text and do not have to edit images. The translated text can be stored separately (for example, in resource files) and extracted when needed, to be combined with the image. Separating text and image processing in this way also makes it possible to use different text styles and sizes for special purposes, such as increasing the size of the text for complex characters such as Kanji, or to adjust visual attributes for people with some visual impairment, for example color blindness.

The Java code and classes supporting UIX Dynamic Images are all located in the `oracle.cabo.image` package and its subpackages. UIX Components depend on UIX Dynamic Images for their own rendered images.

UIX Styles

UIX Styles provide an architecture for defining and customizing stylesheets for different end user environments (for example, locales, browsers, or platforms).

Stylesheets provide a centralized mechanism for defining and altering the appearance of pages separate from the content they contain.

UIX Styles include a new XML Style Sheet Language (XSS) for defining environment-specific stylesheets. XSS is based on Cascading Style Sheets (CSS). UIX Styles also feature server-side APIs for managing style information, including a facility to generate CSS stylesheets dynamically at runtime.

The Java code and classes supporting UIX Styles are all located in the `oracle.cabo.style` package and its subpackages. UIX Components and UIX Dynamic Images depend on UIX Styles for their own style information.

UIX Share

All UIX projects depend on common utility classes provided by UIX Share.

The UIX Share classes include functionality that is useful to all UIX web applications, such as configuration support and localization. The Java code and classes supporting UIX Share are all located in the `oracle.cabo.share` package and its subpackages.

Which UIX Technologies to Use?

The UIX technology stack is open and flexible; you have the choice of using as many of its subproducts as you need. Keep in mind, however, that using some UIX subproducts requires the use of others. For instance, UIX Components use UIX Dynamic Images and UIX Styles to render the images and stylesheets for its pages, respectively, and thus it requires their presence. However, there is no requirement that you use those subproducts in any way beyond UIX Components' own internal usage of them.

It is important to note that the various UIX technologies have been designed to work together. This means that sometimes one UIX project can make it easier to use another. As an example, the UIX Controller will automatically create and cache the pages specified in the UIX language because it has built-in support for this. If you use UIX without UIX Components, you will need to write some code to load in your UIX language document and display it. Conversely, if you use UIX Components without the UIX language, you may have to write some code telling the UIX Framework how to display your own pages. In other words, the whole UIX technology stack is definitely worth more than the sum of its parts!

Here are some recommendations for which technologies to use:

- If you are starting a new web application from scratch.

We recommend you use UIX Controller to manage your application flow and that you use UIX Components and the UIX Language to specify your page layouts and events. This enables you to get the most functionality from UIX with the least work on your part.

- If you cannot replace your existing application flow management technology, but you have flexibility on your page rendering.

  We recommend you use UIX Components and the UIX language to create and render your pages. This enables you to get the advantages of UIX Components (agent-based rendering architecture, high level page beans, localization, and so on) even if you can't use the entire UIX stack. Keep in mind, however, that some of the UIX Controller code might still be useful to your server-side flow management, even if you do not adopt UIX Controller entirely. For example, UIX Controller includes utility code for handling file uploads that is generally useful for Java servlet-based applications.

- If you have existing pages (JSPs or dynamic HTML) that you need to manage through a servlet.

  Consider using the UIX Controller servlet to manage logons, handle errors, and provide other utilities that are missing from the basic servlet architecture. This will also make it easier to include additional pages based on UIX Components later on.

- If you have some existing pages designed with HTML or JSPs, but need to implement new pages for your application.

  We ask that you consider using UIX Components and the UIX language for all your pages. This is possible because UIX Components and the UIX language provide easy ways to intersperse other content such as existing JSPs and HTML on the same page using its passthrough capability. Doing so gives you the opportunity to consolidate your pages on one technology later on and transition as you go.

- If you would like to use UIX Components beans, but are already using Java-based page rendering for other parts of your page(s).

  You can still use UIX Components beans on a page through the Java web bean classes. The generated output can be merged into your existing Java-generated page output. The decision to also use UIX Controller for page management is independent of this choice.

- If you cannot change your current page rendering technology, but you need localizable images in your web application.

Consider using UIX Dynamic Images to generate images including text that are localized.

- If you cannot change your current page rendering technology, but you need stylesheets for your product that are tailored to each viewer's browser, locale, or preferences.

  Consider using UIX Styles to generate and cache individual stylesheets based on variants.

For More Information About UIX

Here are sources of more information about UIX:

See Also: For sample JDeveloper Demonstration code for UIX:

- `http://otn.oracle.com/sample_code/products/jdev/content.html`
- The complete UIX Developer's Guide is included in the JDeveloper online help.

A

XDK for Java: Specifications and Quick References

This appendix describes the XDK for Java specifications and quick references for each XML component for Java. The quick references list the main APIs, classes, and associated methods for each XDK for Java component.

This appendix contains the following sections:

XML Parser for Java Quick Reference

Note: The XML Parser for Java methods are listed in these places:

- *Oracle9i XML API Reference - XDK and Oracle XML DB*
- http://otn.oracle.com/tech/xml
- Your installed software under doc/

XML Parser for Java Specifications

The Oracle XML Parser for Java, Version 2 specifications follow:

- New high performance architecture
- Integrated support for W3C XSLT 1.0 Recommendation
- Supports validation and non-validation modes
- Built-in Error Recovery until fatal error
- Integrated Document Object Model (DOM) Level 1.0 and 2.0 API
- Integrated SAX 1.0 and 2.0 API
- Supports W3C Recommendation for XML Namespaces

Requirements

Operating Systems: Any with Java 1.1.x support

JAVA: JDK 1.1.x. or later.

The contents of both the Windows and UNIX versions are identical. They are simply archived differently for operating system compatibility and your convenience.

Online Documentation

Documentation for Oracle XML Parser for Java is located in the doc/ directory in your install area.

Release Specific Notes

The readme.html file in the root directory of the archive contains release specific information including bug fixes, API additions, and so on.

Oracle XML Parser is an early adopter release and is written in Java. It will check if an XML document is well-formed and, optionally, if it is valid. The parser will construct a Java object tree which can be accessed. It also contains an integrated XSLT processor for transforming XML documents.

Standards Conformance

The parser conforms to the following W3C Recommendations:

- Extensible Markup Language (XML) 1.0
  http://www.w3.org/TR/1998/REC-xml-19980210
- Namespaces in XML at http://www.w3.org/TR/REC-xml-names/
- Document Object Model Level 1 1.0
  http://www.w3.org/TR/REC-DOM-Level-1/
- Document Object Model Level 2
  http://www.w3.org/TR/DOM-Level-2-Core/
- XML Path Language (XPath) 1.0
  http://www.w3.org/TR/1999/REC-xpath-19991116
- XML Transformations (XSLT) 1.0
  http://www.w3.org/TR/1999/REC-xslt-19991116

The parser also conforms to the following W3C *Proposed* Recommendations:

- XML Schema Part 1: Structures http://www.w3.org/TR/xmlschema-1
- XML Schema Part 2: Datatypes http://www.w3.org/TR/xmlschema-2

In addition, the parser implements the following interfaces defined by the XML development community:

- Simple API for XML (SAX) 1.0 and 2.0 at
  http://www.megginson.com/SAX/index.html

Supported Character Set Encodings

The XML Parser for Java currently supports the following encodings:

- BIG 5

- EBCDIC-CP-*
- EUC-JP
- EUC-KR
- GB2312
- ISO-2022-JP
- ISO-2022-KR
- ISO-8859-1to -9
- ISO-10646-UCS-2
- ISO-10646-UCS-4
- KOI8-R
- Shift_JIS
- US-ASCII
- UTF-8
- UTF-16

Default: UTF-8 is the default encoding if none is specified. Any other ASCII or EBCDIC based encodings that are supported by the JDK may be used.However, they must be specified in the format required by the JDK instead of as official character set names defined by IANA.

Error Recovery

The parser also provides error recovery. It will recover from most errors and continue processing until a fatal error is encountered.

XDK for Java: XML Schema Processor

See Also:

- Chapter 6, "XML Schema Processor for Java"
- The readme.txt file in your installed software's doc/ directory. This software can also be downloaded from http://otn.oracle.com/tech/xml

XDK for Java: XML Class Generator for Java

Oracle XML Class Generator for Java requires Oracle XML Parser for Java. The XML Document, printed by the generated classes, confirms to the W3C recommendation for Extensible Markup Language (XML) 1.0. Oracle XML Class Generator can optionally generate validating Java source files. It also optionally generates Javadoc comments in the source files.

Oracle XML Class Generator supports the following encodings for printing the XMLDocument:

UTF-8, UTF-16, ISO-10646-UCS-2, ISO-10646-UCS-4, US-ASCII, EBCDIC-CP-US, ISO-8859-1, and Shift_SJIS.

ASCII is the default encoding if none is specified. Any other ASCII or EBCDIC based encodings that are supported by the JDK can be used.

XDK for Java: XSQL Servlet

Downloading and Installing XSQL Servlet

Downloading XSQL Servlet from OTN

You can download XSQL Servlet distribution from:

http://otn.oracle.com/tech/xml/xsql_servlet

1. Click the 'Software' icon at the top of the page:
2. Log in with your OTN username and password (registration is free if you do not already have an account).

3. Selecting whether you want the NT or Unix download (both contain the same files)
4. Acknowledge the licensing agreement and download survey
5. Clicking on xsqlservlet_v1.0.2.0.tar.gz or xsqlservlet_v1.0.2.0.zip

Extracting the Files in the Distribution

To extract the contents of XSQL Servlet distribution, do the following:

1. Choose a directory under which you would like the .\xsql directory and subdirectories to go, for example, C:\
2. Change directory to C:\, then extract the XSQL downloaded archive file there. For example:

   UNIX:

   ```
   tar xvfz xsqlservlet_v1.0.2.0.tar.gz
   ```

   Windows NT:

   ```
   pkzip25 -extract -directories xsqlservlet_v1.0.2.0.zip
   ```

   using the pkzip25 command-line tool or the WinZip visual archive extraction tool.

Windows NT: Starting the Web-to-Go Server

XSQL Servlet comes bundled with the Oracle Web-to-go server that is pre-configured to use XSQL Pages. The Web-to-go web server is a single-user server, supporting the Servlet 2.1 API, used for mobile application deployment and for development. This is a great way to try XSQL Pages out on your Windows machine before delving into the details of configuring another Servlet Engine to run XSQL Pages.

> **Note:** The Web-to-go Web server is part of Oracle's development and deployment platform for mobile applications. For more information on Web-to-go, see http://www.oracle.com/mobile.

Windows NT users can get started quickly with XSQL Pages by following these steps:

1. Running the xsql-wtg.bat script in the .\xsql directory.

2. Browsing the URL http://localhost:7070/xsql/index.html

If you get an error starting this script, edit the xsql-wtg.bat file to properly set the two environment variables JAVA and XSQL_HOME to appropriate values for your machine.

```
REM ------------------------------------------------
REM Set the 'JAVA' variable equal to the full path
REM of your Java executable.
REM ------------------------------------------------
set JAVA=J:\java1.2\jre\bin\java.exe
set XSQL_HOME=C:\xsql
REM ------------------------------------------------
REM Set the 'XSQL_HOME' variable equal to the full
REM path of where you install the XSQL Servlet
REM distribution.
REM ------------------------------------------------
```

Then, repeat the two preceding steps.

If you get an error connecting to the database when you try the demos, you'll need to go on to the next section, then try the preceding steps again after setting up your database connection information correctly in the XSQLConfig.xml file.

Setting Up the Database Connection Definitions for Your Environment

The demos are set up to use the SCOTT schema on a database on your local machine (that is, the machine where the web server is running). If you are running a local database and have a SCOTT account whose password is TIGER, then you are all set. Otherwise, you need to edit the `.\xsql\lib\XSQLConfig.xml` file to correspond to your appropriate values for username, password, dburl, and driver values for the connection named "demo".

```
<?xml version="1.0" ?>
    <XSQLConfig>
         :
      <connectiondefs>
        <connection name="demo">
          <username>scott</username>
          <password>tiger</password>
          <dburl>jdbc:oracle:thin:@localhost:1521:ORCL</dburl>
          <driver>oracle.jdbc.driver.OracleDriver</driver>
        </connection>
        <connection name="lite">
          <username>system</username>
```

```
            <password>manager</password>
            <dburl>jdbc:Polite:POlite</dburl>
            <driver>oracle.lite.poljdbc.POLJDBCDriver</driver>
          </connection>
        </connectiondefs>
            :
</XSQLConfig>
```

UNIX: Setting Up Your Servlet Engine to Run XSQL Pages

UNIX users and any user wanting to install the XSQL Servlet on other web servers should continue with the instructions below depending on the web server you're trying to use. In every case, there are 3 basic steps:

1. Include the list of XSQL Java archives as well as the directory where XSQLConfig.xml resides (by default ./xsql/lib) in the server CLASSPATH.

   Note: For convenience, the xsqlservlet_v1.0.2.0.tar.gz and xsqlservlet_v1.0.2.0.zip distributions include the .jar files for the Oracle XML Parser for Java (V2), the Oracle XML SQL Utilities for Java, and the 8.1.6 JDBC driver in the .\lib subdirectory, along with Oracle XSQL Pages' own .jar archive.

2. Map the .xsql file extension to the oracle.xml.xsql.XSQLServlet servlet class
3. Map a virtual directory /xsql to the directory where you extracted the XSQL files (to access the on-line help and demos)

XSQL Servlet Specifications

The following lists the XSQL servlet specifications:

- Produce dynamic XML documents based on one or more SQL queries
- Optionally transforms the resulting XML document in the server or client using XSLT
- Supports W3C XML 1.0 Recommendation
- Supports Document Object Model (DOM) Level 1.0 and 2.0 API
- Support the W3C XSLT 1.0 Recommendation
- Supports W3C Recommendation for XML Namespaces

Character Set Support

XSQL Servlet supports the following character set encodings:

- BIG
- EBCDIC-CP-*
- EUC-JP
- EUC-KR
- GB2312
- ISO-2022-JP
- ISO-2022-KR
- ISO-8859-1to -9
- ISO-10646-UCS-2
- ISO-10646-UCS-4
- KOI8-R
- Shift_JIS
- US-ASCII
- UTF-8
- UTF-16

B

XDK for PL/SQL: Specifications

This Appendix describes Oracle XDK for PL/SQL specifications. It contains the following sections:

- XML Parser for PL/SQL
- XML Parser for PL/SQL Specifications

XML Parser for PL/SQL

XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constraints on the storage layout and logical structure.

A software module called an XML processor is used to read XML documents and provide access to their content and structure. It is assumed that an XML processor is doing its work on behalf of another module, called the application.

Oracle XML Parser Features

The XML Parser for PL/SQL parses an XML document (or a standalone DTD) so that it can be processed by an application. Library and command-line versions are provided supporting the following standards and features:

- DOM (Document Object Model) support is provided compliant with the W3C DOM 1.0 Recommendation. These APIs permit applications to access and manipulate an XML document as a tree structure in memory. This interface is used by such applications as editors.
- SAX (Simple API for XML) support is also provided compliant with the SAX 1.0 specification. These APIs permit an application to process XML documents using an event-driven model.
- Support is also included for XML Namespaces 1.0 thereby avoiding name collisions, increasing reusability and easing application integration.
- Able to run on Oracle9*i* and Oracle9i Application Server.
- C and C++ versions initially available for Windows, Solaris, and Linux.

Additional features include:

- Validating and non-validating operation modes
- Built-in error recovery until fatal error
- DOM extension APIs for document creation Oracle XSL-Transform Processors

Version 2 of the Oracle XML Parsers include an integrated XSL-Transformation (XSL-T) Processor for transforming XML data using XSL stylesheets. Using the XSL-T processor, you can transform XML documents from XML to XML, HTML, or virtually any other text-based format. These processors support the following standards and features:

- Compliant with the W3C XSL Transform Proposed Recommendation 1.0
- Compliant with the W3C XPath Proposed Recommendation 1.0
- Integrated into the XML Parser for improved performance and scalability
- Available with library and command-line interfaces for Java, C, C++, and PL/SQL

Namespace Support

The Java, C, and C++ parsers also support XML Namespaces. Namespaces are a mechanism to resolve or avoid name collisions between element types (tags) or attributes in XML documents. This mechanism provides "universal" namespace element types and attribute names whose scope extends beyond the containing document. Such tags are qualified by uniform resource identifiers (URIs), such as <oracle:EMP xmlns:oracle="http://www.oracle.com/xml"/>. For example, namespaces can be used to identify an Oracle <EMP> data element as distinct from another company's definition of an <EMP> data element. This enables an application to more easily identify elements and attributes it is designed to process. The Java, C, and C++ parsers support namespaces by being able to recognize and parse universal element types and attribute names, as well as unqualified "local" element types and attribute names.

Validating and Non-Validating Mode Support

The Java, C, and C++ parsers can parse XML in validating or non-validating modes. In non-validating mode, the parser verifies that the XML is well-formed and parses the data into a tree of objects that can be manipulated by the DOM API. In validating mode, the parser verifies that the XML is well-formed and validates the XML data against the DTD (if any). Validation involves checking whether or not the attribute names and element tags are legal, whether nested elements belong where they are, and so on.

Example Code

See Chapter 20, "XML Parser for PL/SQL" for example code and suggestions on how to use the XML Parsers.

IXML Parser for PL/SQL Directory Structure

The following lists the XML Parser for PL/SQL directory structure in $ORACLE_HOME/xdk/plsql/parser:

- Windows NT
  - license.html - copy of license agreement
  - readme.html - release and installation notes
  - doc\ - directory for parser apis.
  - lib\ - directory for parser sql and class files
  - sample\ - sample code
- UNIX
  - license.html — copy of license agreement
  - readme.html — release and installation notes
  - doc/ — directory for parser apis
  - lib/ — directory for parser sql and class files
  - sample/ — sample code files

DOM and SAX APIs

XML APIs generally fall into two categories: event-based and tree-based. An event-based API (such as SAX) uses callbacks to report parsing events to the application. The application deals with these events through customized event handlers. Events include the start and end of elements and characters. Unlike tree-based APIs, event-based APIs usually do not build in-memory tree representations of the XML documents. Therefore, in general, SAX is useful for applications that do not need to manipulate the XML tree, such as search operations, among others. For example, the following XML document:

```
<?xml version="1.0"?>
  <EMPLIST>
    <EMP>
     <ENAME>MARTIN</ENAME>
    </EMP>
    <EMP>
     <ENAME>SCOTT</ENAME>
    </EMP>
  </EMPLIST>
```

Becomes a series of linear events:

```
start document
start element: EMPLIST
```

```
start element: EMP
start element: ENAME
characters: MARTIN
end element: EMP
start element: EMP
start element: ENAME
characters: SCOTT
end element: EMP
end element: EMPLIST
end document
```

A tree-based API (such as DOM) builds an in-memory tree representation of the XML document. It provides classes and methods for an application to navigate and process the tree. In general, the DOM interface is most useful for structural manipulations of the XML tree, such as reordering elements, adding or deleting elements and attributes, renaming elements, and so on.

XML Parser for PL/SQL Specifications

These are the Oracle XML Parser for PL/SQL specifications:

- Supports validation and non-validation modes
- Includes built-in error recovery until fatal error
- Supports the W3C XML 1.0 Recommendation
- Supports the W3C XSL-T Final Working Draft

This PL/SQL implementation of the XML processor (or parser) follows the W3C XML specification (rev REC-xml-19980210) and included the required behavior of an XML processor in terms of how it must read XML data and the information it must provide to the application.

XML Parser for PL/SQL: Default Behavior

The following is the default behavior for this PLSQL XML parser:

- A parse tree which can be accessed by DOM APIs is built
- The parser is validating if a DTD is found, otherwise it is non-validating
- Errors are not recorded unless an error log is specified; however, an application error will be raised if parsing fails

The types and methods described in this document are made available by the PLSQL package xmlparser.

- Integrated Document Object Model (DOM) Level 1.0 API

Supported Character Set Encodings

Supports documents in the following Oracle database encodings:

- BIG 5
- EBCDIC-CP-*
- EUC-JP
- EUC-KR
- GB2312
- ISO-2022-JP
- ISO-2022-KR
- ISO-8859-1to -9
- KOI8-R
- Shift_JIS
- US-ASCII
- UTF-8

Default: UTF-8 is the default encoding if none is specified. Any other ASCII or EBCDIC based encodings that are supported by the Oracle 9i database may be used.

Requirements

Oracle*9i* database with the Java option enabled.

Online Documentation

Documentation for Oracle XML Parser for PL/SQL is located in the doc directory in your install area and also in *Oracle9i XML API Reference - XDK and Oracle XML DB.*

Release Specific Notes

The Oracle XML parser for PL/SQL is an early adopter release and is written in PL/SQL and Java. It will check if an XML document is well-formed and, optionally, if it is valid. The parser will construct an object tree which can be accessed through PL/SQL interfaces.

Standards Conformance

The parser conforms to the following standards:

- W3C recommendation for Extensible Markup Language (XML) 1.0 at http://www.w3.org/TR/1998/REC-xml-19980210
- W3C recommendation for Document Object Model Level 1 1.0 at http://www.w3.org/TR/REC-DOM-Level-1/

The parser currently does not currently have SAX or Namespace support. These will be made available in a future version.

Error Recovery

The parser also provides error recovery. It will recover from most errors and continue processing until a fatal error is encountered.

Important note: The contents of both the Windows and UNIX versions are identical. They are simply archived differently for operating system compatibility and your convenience.

See Also:

- *Oracle9i XML API Reference - XDK and Oracle XML DB*
- Chapter 8, "XML SQL Utility (XSU)"
- `http://otn.oracle.com/tech/xml`

Glossary

access control entry (ACE)

An entry in the access control list that grants or denies access to a given principal.

access control list (ACL)

A list of access control entries that determines which principals have access to a given resource or resources.

ACE

Access Control Entry. See access control entry.

ACL

Access Control List. See access control list.

API

Application Program Interface. See application program interface.

application program interface (API)

A set of public programmatic interfaces that consist of a language and message format to communicate with an operating system or other programmatic environment, such as databases, Web servers, JVMs, and so forth. These messages typically call functions and methods available for application development.

application server

A server designed to host applications and their environments, permitting server applications to run. A typical example is OAS, which is able to host Java, C, C++, and PL/SQL applications in cases where a remote client controls the interface. See also Oracle Application Server.

attribute

A property of an element that consists of a name and a value separated by an equals sign and contained within the start-tags after the element name. In this example, `<Price units='USD'>5</Price>`, units is the attribute and USD is its value, which must be in single or double quotes. Attributes may reside in the document or DTD. Elements may have many attributes but their retrieval order is not defined.

BC4J

Business Components for Java, a J2EE application development framework that comes with JDeveloper. BC4J is an object-relational mapping tool that implements J2EE Design Patterns.

BFILES

External binary files that exist outside the database tablespaces residing in the operating system. BFILES are referenced from the database semantics, and are also known as External LOBs.

Binary Large Object (BLOB)

A Large Object datatype whose content consists of binary data. Additionally, this data is considered raw as its structure is not recognized by the database.

BLOB

See Binary Large Object.

Business-to-Business (B2B)

A term describing the communication between businesses in the selling of goods and services to each other. The software infrastructure to enable this is referred to as an exchange.

Business-to-Consumer (B2C)

A term describing the communication between businesses and consumers in the selling of goods and services.

callback

A programmatic technique in which one process starts another and then continues. The second process then calls the first as a result of an action, value, or other event. This technique is used in most programs that have a user interface to allow continuous interaction.

cartridge

A stored program in Java or PL/SQL that adds the necessary functionality for the database to understand and manipulate a new datatype. Cartridges interface through the Extensibility Framework within Oracle 8 or later. Oracle Text is such a cartridge, adding support for reading, writing, and searching text documents stored within the database.

Cascading Style Sheets

A simple mechanism for adding style (fonts, colors, spacing, and so on) to Web documents.

CDATA

See character data.

CDF

Channel Definition Format. Provides a way to exchange information about channels on the internet.

CGI

See Common Gateway Interface.

character data (CDATA)

Text in a document that should not be parsed is put within a CDATA section. This allows for the inclusion of characters that would otherwise have special functions, such as &, <, >, and so on. CDATA sections can be used in the content of an element or in attributes.

child element

An element that is wholly contained within another, which is referred to as its parent element. For example <Parent><Child></Child></Parent> illustrates a child element nested within its parent element.

Class Generator

A utility that accepts an input file and creates a set of output classes that have corresponding functionality. In the case of the XML Class Generator, the input file is a DTD and the output is a series of classes that can be used to create XML documents conforming with the DTD.

CLASSPATH

The operating system environmental variable that the JVM uses to find the classes it needs to run applications.

client/server

The term used to describe the application architecture where the actual application runs on the client but accesses data or other external processes on a server across a network.

Character Large Object (CLOB)

The LOB datatype whose value is composed of character data corresponding to the database character set. A CLOB may be indexed and searched by the Oracle Text search engine.

CLOB

See Character Large Object.

command line

The interface method in which the user enters in commands at the command interpreter's prompt.

Common Gateway Interface (CGI)

The programming interfaces enabling Web servers to execute other programs and pass their output to HTML pages, graphics, audio, and video sent to browsers.

Common Object Request Broker API (CORBA)

An Object Management Group standard for communicating between distributed objects across a network. These self-contained software modules can be used by applications running on different platforms or operating systems. CORBA objects and their data formats and functions are defined in the Interface Definition Language (IDL), which can be compiled in a variety of languages including Java, C, C++, Smalltalk and COBOL.

Common Oracle Runtime Environment (CORE)

The library of functions written in C that provides developers the ability to create code that can be easily ported to virtually any platform and operating system.

Content

The body of a resource is what you get when you treat the resource like a file and ask for its contents. Content is always an XMLType.

CORBA

See Common Object Request Broker API.

CSS

See Cascading Style Sheets.

Database Access Descriptor (DAD)

A DAD is a named set of configuration values used for database access. A DAD specifies information such as the database name or the Oracle Net service name, the `ORACLE_HOME` directory, and Globalization Support configuration information such as language, sort type, and date language.

datagram

A text fragment, which may be in XML format, that is returned to the requester embedded in an HTML page from a SQL query processed by the XSQL Servlet.

DBURITYPE

The Oracle9*i* datatype used for storing instances of the datatype that permits XPath-based navigation of database schemas.

DOCTYPE

The term used as the tag name designating the DTD or its reference within an XML document. For example, `<!DOCTYPE person SYSTEM "person.dtd">` declares the root element name as person and an external DTD as person.dtd in the file system. Internal DTDs are declared within the DOCTYPE declaration.

Document Object Model (DOM)

An in-memory tree-based object representation of an XML document that enables programmatic access to its elements and attributes. The DOM object and its interface is a W3C recommendation. It specifies the Document Object Model of an XML Document including the APIs for programmatic access. DOM views the parsed document as a tree of objects.

Document Type Definition (DTD)

A set of rules that define the allowable structure of an XML document. DTDs are text files that derive their format from SGML and can either be included in an XML document by using the DOCTYPE element or by using an external file through a DOCTYPE reference.

DOM

See Document Object Model.

DOM fidelity

To assure the integrity and accuracy of this data, for example, when regenerating XML documents stored in Oracle XML DB, Oracle XML DB uses a data integrity mechanism, called DOM fidelity. DOM fidelity refers to when the returned XML documents are identical to the original XML document, particularly for purposes of DOM traversals. Oracle XML DB assures DOM fidelity by using a binary attribute, SYS_XDBPD$.

DTD

See Document Type Definition.

EDI

Electronic Data Interchange.

element

The basic logical unit of an XML document that can serve as a container for other elements such as children, data, and attributes and their values. Elements are identified by start-tags, such as <name>, and end-tags, such as </name>, or in the case of empty elements, <name/>.

empty element

An element without text content or child elements. It can only contain attributes and their values. Empty elements are of the form <name/> or <name></name>, where there is no space between the tags.

Enterprise Java Bean (EJB)

An independent program module that runs within a JVM on the server. CORBA provides the infrastructure for EJBs, and a container layer provides security, transaction support, and other common functions on any supported server.

empty element

An element without text content or child elements. It may only contain attributes and their values. Empty elements are of the form <name/> or <name></name> where there is no space between the tags.

entity

A string of characters that may represent either another string of characters or special characters that are not part of the document's character set. Entities and the text that is substituted for them by the parser are declared in the DTD.

existnode

The SQL operator that returns a TRUE or FALSE based upon the existence of an `XPath` within an `XMLType`.

eXtensible Markup Language (XML)

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.

eXtensible Stylesheet Language (XSL)

The language used within stylesheets to transform or render XML documents. There are two W3C recommendations covering XSL stylesheets—XSL Transformations (XSLT) and XSL Formatting Objects (XSLFO).

(W3C) eXtensible Stylesheet Language. XSL consists of two W3C recommendations: XSL Transformations for transforming one XML document into another and XSL Formatting Objects for specifying the presentation of an XML document. XSL is a language for expressing stylesheets. It consists of two parts:

- A language for transforming XML documents (XSLT), and
- An XML vocabulary for specifying formatting semantics (XSLFO).

An XSL stylesheet specifies the presentation of a class of XML documents by describing how an instance of the class is transformed into an XML document that uses the formatting vocabulary.

eXtensible Stylesheet Language Formatting Object (XSLFO)

The W3C standard specification that defines an XML vocabulary for specifying formatting semantics. See FOP.

eXtensible Stylesheet Language Transformation (XSLT)

Also written as XSL-T. The XSL W3C standard specification that defines a transformation language to convert one XML document into another.

extract

The SQL operator that retrieves fragments of XML documents stored as `XMLType`.

Folder

A directory or node in the Oracle XML DB repository that contains or can contain a resource. A folder is also a resource.

Foldering

A feature in Oracle XML DB that allows content to be stored in a hierarchical structure of resources.

FOP

Print formatter driven by XSL formatting objects. It is a Java application that reads a formatting object tree and then renders the resulting pages to a specified output. Output formats currently supported are PDF, PCL, PS, SVG, XML (area tree representation), Print, AWT, MIF and TXT. The primary output target is PDF.

functional index

A database index that, when created, permits the results of known queries to be returned much more quickly.

HASPATH

The SQL operator that is part of Oracle Text and used for querying XMLType datatypes for the existence of a specific XPath.

hierarchical indexing

The data relating a folder to its children is managed by the Oracle XML DB hierarchical index, which provides a fast mechanism for evaluating path names similar to the directory mechanisms used by operating system filesystems. Any pathname-based access will normally use the Oracle XML DB hierarchical index.

HTML

See Hypertext Markup Language.

HTTP

See Hypertext Transport Protocol.

HTTPURITYPE

The datatype used for storing instances of the datatype that permits XPath-based navigation of database schemas in remote databases.

hypertext

The method of creating and publishing text documents in which users can navigate between other documents or graphics by selecting words or phrases designated as hyperlinks.

Hypertext Markup Language (HTML)

The markup language used to create the files sent to Web browsers and that serves as the basis of the World Wide Web. The next version of HTML will be called xHTML and will be an XML application.

Hypertext Transport Protocol (HTTP)

The protocol used for transporting HTML files across the Internet between Web servers and browsers.

iAS

See Oracle9iAS.

IDE

See Integrated Development Environment.

*i*FS

See Internet File System.

INPATH

The SQL operator that is part of Oracle Text and is used for querying XMLType datatypes for searching for specific text within a specific XPath.

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)

A set of programs designed to aide in the development of software run from a single user interface. JDeveloper is an IDE for Java development as it includes an editor, compiler, debugger, syntax checker, help system, and so on, to permit Java software development through a single user interface.

interMedia

The collection of complex datatypes and their access in Oracle. These include text, video, time-series, and spatial data.

Internet File System (*i*FS)

The Oracle file system and Java-based development environment that either runs inside the database or on a middle tier and provides a means of creating, storing, and managing multiple types of documents in a single database repository.

Internet Inter-ORB Protocol (IIOP)

The protocol used by CORBA to exchange messages on a TCP/IP network such as the Internet.

J2EE

See Java 2 Platform, Enterprise Edition.

Java

A high-level programming language developed and maintained by Sun Microsystems where applications run in a virtual machine known as a JVM. The JVM is responsible for all interfaces to the operating system. This architecture permits developers to create Java applications and applets that can run on any operating system or platform that has a JVM.

Java 2 Platform, Enterprise Edition (J2EE)

The Java platform (Sun Microsystems) that defines multi-tier enterprise computing.

Java API for XML Processing (JAXP)

Enables applications to parse and transform XML documents using an API that is independent of a particular XML processor implementation.

JavaBean

An independent program module that runs within a JVM, typically for creating user interfaces on the client. Also known as Java Bean. The server equivalent is called an Enterprise JavaBean (EJB). See also Enterprise JavaBean.

Java Database Connectivity (JDBC)

The programming API that enables Java applications to access a database through the SQL language. JDBC drivers are written in Java for platform independence but are specific to each database.

Java Developer's Kit (JDK)

The collection of Java classes, runtime, compiler, debugger, and usually source code for a version of Java that makes up a Java development environment. JDKs are designated by versions, and Java 2 is used to designate versions from 1.2 onward.

Java Naming and Directory Interface

A programming interface from Sun for connecting Java programs to naming and directory services such as DNS, LDAP and NDS. Oracle XML DB Resource API for Java / JNDI supports JNDI.

Java Runtime Environment (JRE)

The collection of complied classes that make up the Java virtual machine on a platform. JREs are designated by versions, and Java 2 is used to designate versions from 1.2 onward.

Java Server Page (JSP)

An extension to the servlet functionality that enables a simple programmatic interface to Web pages. JSPs are HTML pages with special tags and embedded Java code that is executed on the Web server or application server providing dynamic functionality to HTML pages. JSPs are actually compiled into servlets when first requested and run in the server's JVM.

Java Virtual Machine (JVM)

The Java interpreter that converts the compiled Java bytecode into the machine language of the platform and runs it. JVMs can run on a client, in a browser, in a middle tier, on an intranet, on an application server such as Oracle9iAS, or in a database server such as Oracle.

JAXP

See Java API for XML Processing.

JDBC

See Java Database Connectivity.

JDeveloper

Oracle's Java IDE that enables application, applet, and servlet development and includes an editor, compiler, debugger, syntax checker, help system, an integrated UML class modeler, and so on. JDeveloper has been enhanced to support XML-based development by including the Oracle XDK for Java, integrated for easy use along with XML support, in its editor.

JDK

See Java Developer's Kit.

JNDI

JServer

The Java Virtual Machine that runs within the memory space of the Oracle database. In Oracle 8i Release 1 the JVM was Java 1.1 compatible while Release 2 is Java 1.2 compatible.

JVM

See Java virtual machine.

LAN

See local area network.

Large Object (LOB)

The class of SQL data type that is further divided into Internal LOBs and External LOBs. Internal LOBs include BLOBs, CLOBS, and NCLOBs while External LOBs include BFILES. See also BFILES, Binary Large Object, Character Large Object.

lazy type conversions

A mechanism used by Oracle XML DB to only convert the XML data for Java when the Java application first asks for it. This saves typical type conversion bottlenecks with JDBC.

listener

A separate application process that monitors the input process.

LOB

See Large Object.

local area network (LAN)

A computer communication network that serves users within a restricted geographical area. LANs consist of servers, workstations, communications hardware (routers, bridges, network cards, and so on) and a network operating system.

name-level locking

Oracle XML DB provides for name-level locking rather than collection-level locking. When a name is added to a collection, an exclusive write lock is not placed on the collection, only that name within the collection is locked. The name modification is put on a queue, and the collection is locked and modified only at commit time.

namespace

The term to describe a set of related element names or attributes within an XML document. The namespace syntax and its usage is defined by a W3C Recommendation. For example, the <xsl:apply-templates / > element is identified as part of the XSL namespace. Namespaces are declared in the XML document or DTD before they are used be using the following attribute syntax: `xmlns:xsl="http://www.w3.org/TR/WD-xsl".`

national Character Large Object (NCLOB)

The LOB datatype whose value is composed of character data corresponding to the database national character set.

NCLOB

See National Character Large Object.

node

In XML, the term used to denote each addressable entity in the 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.

N-tier

The designation for a computer communication network architecture that consists of one or more tiers made up of clients and servers. Typically two-tier systems are made up of one client level and one server level. A three-tier system utilizes two server tiers, typically a database server as one and a Web or application server along with a client tier.

OAG

Open Applications Group.

OAI

Oracle Applications Integrator. Runtime with Oracle iStudio development tool that provides a way for CRM applications to integrate with other ERP systems besides Oracle ERP. Specific APIs must be "message-enabled." It uses standard extensibility hooks to generate or parse XML streams exchanged with other application systems. In development.

OASIS

See Organization for the Advancement of Structured Information.

Object View

A tailored presentation of the data contained in one or more object tables or other views. The output of an Object View query is treated as a table. Object Views can be used in most places where a table is used.

object-relational

The term to describe a relational database system that can also store and manipulate higher-order data types, such as text documents, audio, video files, and user-defined objects.

Object Request Broker (ORB)

Software that manages message communication between requesting programs on clients and between objects on servers. ORBs pass the action request and its parameters to the object and return the results back. Common implementations are JCORB and EJBs. See also CORBA.

OCT

See Ordered Collection in Tables.

OC4J

Oracle9iAS Containers for J2EE, a J2EE deployment tool that comes with JDeveloper.

OE

Oracle Exchange.

OIS

See Oracle Integration Server.

Oracle9iAS (iAS)

The Oracle application server that integrates all the core services and features required for building, deploying, and managing high-performance, n-tier, transaction-oriented Web applications within an open standards framework.

Oracle Integration Server (OIS)

The Oracle product that serves as the messaging hub for application integration. OIS contains an Oracle 8i database with AQ and Oracle Workflow and interfaces to applications using Oracle Message Broker to transport XML-formatted messages between them.

ORACLE_HOME

The operating system environmental variable that identifies the location of the Oracle database installation for use by applications.

Ordered Collection in Tables (OCT)

When elements of a VARRAY are stored in a separate table, they are referred to as an Ordered Collection in Tables.

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 XML DB

A high-performance XML storage and retrieval technology provided with Oracle database server. It is based on the W3C XML data model.

ORB

See Object Request Broker.

Organization for the Advancement of Structured Information (OASIS)

An organization of members chartered with promoting public information standards through conferences, seminars, exhibits, and other educational events. XML is a standard that OASIS is actively promoting as it is doing with SGML.

parent element

An element that surrounds another element, which is referred to as its child element. For example, <Parent><Child></Child></Parent> illustrates a parent element wrapping its child element.

parser

In XML, a software program that accepts as input an XML document and determines whether it is well-formed and, optionally, valid. The Oracle XML Parser supports both SAX and DOM interfaces.

Parsed Character Data (PCDATA)

The element content consisting of text that should be parsed but is not part of a tag or nonparsed data.

pathname

The name of a resource that reflects its location in the repository hierarchy. A pathname is composed of a root element (the first /), element separators (/) and various sub-elements (or path elements). A path element may be composed of any character in the database character set except ("\", "/"). These characters have a special meaning for Oracle XML DB. Forward slash is the default name separator in a path name and backward slash may be used to escape characters.

PCDATA

See Parsed Character Data.

PDA

Personal Digital Assistant, such as a Palm Pilot.

PL/SQL

The Oracle procedural database language that extends SQL. It is used to create programs that can be run within the database.

principal

An entity that may be granted access control privileges to an Oracle XML DB resource. Oracle XML DB supports as principals:

- Database users.
- Database roles. A database role can be understood as a group, for example, the DBA role represents the DBA group of all the users granted the DBA role.

Users and roles imported from an LDAP server are also supported as a part of the database's general authentication model.

prolog

The opening part of an XML document containing the XML declaration and any DTD or other declarations needed to process the document.

PUBLIC

The term used to specify the location on the Internet of the reference that follows.

RDF

Resource Definition Framework.

renderer

A software processor that outputs a document in a specified format.

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 pathname.

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 UTF8 character set (NVARCHAR).

result set

The output of a SQL query consisting of one or more rows of data.

root element

The element that encloses all the other elements in an XML document and is between the optional prolog and epilog. An XML document is only permitted to have one root element.

SAX

See Simple API for XML.

schema

The definition of the structure and data types within a database. It can also be used to refer to an XML document that support the XML Schema W3C recommendation.

Secure Sockets Layer (SSL)

The primary security protocol on the Internet; it utilizes a public key / private key form of encryption between browsers and servers.

Server-Side Include (SSI)

The HTML command used to place data or other content into a Web page before sending it to the requesting browser.

servlet

A Java application that runs in a server, typically a Web or application server, and performs processing on that server. Servlets are the Java equivalent to CGI scripts.

session

The active connection between two tiers.

SGML

See Structured Generalized Markup Language.

Simple API for XML (SAX)

An XML standard interface provided by XML parsers and used by event-based applications.

Simple Object Access Protocol (SOAP)

An XML-based protocol for exchanging information in a decentralized, distributed environment.

SOAP

See Simple Object Access Protocol.

SQL

See Structured Query Language.

SSI

See Server-side Include.

SSL

See Secure Sockets Layer.

Structured Generalized Markup Language (SGML)

An ISO standard for defining the format of a text document implemented using markup and DTDs.

Structured Query Language (SQL)

The standard language used to access and process data in a relational database.

Stylesheet

In XML, the term used to describe an XML document that consists of XSL processing instructions used by an XSL processor to transform or format an input XML document into an output one.

SYSTEM

Specifies the location on the host operating system of the reference that follows.

SYS_XMLAGG

The term used to specify the location on the host operating system of the reference that follows.

SYS_XMLGEN

The native SQL function that returns as an XML document the results of a passed-in SQKL query. This can also be used to instantiate an XMLType.

tag

A single piece of XML markup that delimits the start or end of an element. Tags start with < and end with >. In XML, there are start-tags (<name>), end-tags (</name>), and empty tags (<name/>).

TCP/IP

See Transmission Control Protocol/Internet Protocol.

thread

In programming, a single message or process execution path within an operating system that supports concurrent execution (multithreading).

Transmission Control Protocol/Internet Protocol (TCP/IP)

The communications network protocol that consists of the TCP which controls the transport functions and IP which provides the routing mechanism. It is the standard for Internet communications.

Transviewer

The Oracle term used to describe the Oracle XML JavaBeans included in the XDK for Java.

TransXUtility

TransXUtility is a Java API that simplifies the loading of translated seed data and messages into a database.

UDDI

See Universal Description, Discovery and Integration.

UIX

See User Interface XML.

Uniform Resource Identifier (URI)

The address syntax that is used to create URLs and XPaths.

Uniform Resource Locator (URL)

The address that defines the location and route to a file on the Internet. URLs are used by browsers to navigate the World Wide Web and consist of a protocol prefix, port number, domain name, directory and subdirectory names, and the file name. For example http://technet.oracle.com:80/tech/xml/index.htm specifies the location and path a browser will travel to find OTN's XML site on the World Wide Web.

Universal Description, Discovery and Integration (UDDI)

This specification provides a platform-independent framework using XML to describe services, discover businesses, and integrate business services on the Internet.

URI

See Uniform Resource Identifier.

URL

See Uniform Resource Locator.

user interface (UI)

The combination of menus, screens, keyboard commands, mouse clicks, and command language that defines how a user interacts with a software application.

User Interface XML (UIX)

A set of technologies that constitute a framework for building web applications.

valid

The term used to refer to an XML document when its structure and element content is consistent with that declared in its referenced or included DTD.

W3C

See World Wide Web Consortium (W3C).

WAN

See wide area network.

WebDAV

See World Wide Web distributed authoring and versioning.

Web Request Broker (WRB)

The cartridge within OAS that processes URLs and sends them to the appropriate cartridge.

Web Services Description Language (WSDL)

A general purpose XML language for describing the interface, protocol bindings, and deployment details of Web services.

well-formed

The term used to refer to an XML document that conforms to the syntax of the XML version declared in its XML declaration. This includes having a single root element, properly nested tags, and so forth.

wide area network (WAN)

A computer communication network that serves users within a wide geographic area, such as a state or country. WANs consist of servers, workstations,

communications hardware (routers, bridges, network cards, and so on), and a network operating system.

Working Group (WG)

The committee within the W3C that is made up of industry members that implement the recommendation process in specific Internet technology areas.

World Wide Web Consortium (W3C)

An international industry consortium started in 1994 to develop standards for the World Wide Web. It is located at www.w3c.org.

World Wide Web Distributed Authoring and Versioning (WebDAV)

The Internet Engineering Task Force (IETF) standard for collaborative authoring on the Web. Oracle XML DB Foldering and Security features are WebDAV-compliant.

Wrapper

The term describing a data structure or software that wraps around other data or software, typically to provide a generic or object interface.

WSDL

See Web Services Description Language.

XDBbinary

An XML element defined by the Oracle XML DB schema that contains binary data. XDBbinary elements are stored in the repository when completely unstructured binary data is uploaded into Oracle XML DB.

XDK

See XML Developer's Kit.

XLink

The XML Linking language consisting of the rules governing the use of hyperlinks in XML documents. These rules are being developed by the XML Linking Group under the W3C recommendation process. This is one of the three languages XML supports to manage document presentation and hyperlinks (XLink, XPointer, and XPath).

XML

See eXtensible Markup Language.

XML Developer's Kit (XDK)

The set of libraries, components, and utilities that provide software developers with the standards-based functionality to XML-enable their applications. In the case of the Oracle XDK for Java, the kit contains an XML parser, an XSLT processor, the XML Class Generator, the Transviewer JavaBeans, and the XSQL Servlet.

XML Gateway

A set of services that allows for easy integration with the Oracle e-Business Suite to create and consume XML messages triggered by business events.

XML Query

The W3C's effort to create a standard for the language and syntax to query XML documents.

XML Schema

The W3C's effort to create a standard to express simple data types and complex structures within an XML document. It addresses areas currently lacking in DTDs, including the definition and validation of data types. Oracle XML Schema Processor automatically ensures validity of XML documents and data used in e-business applications, including online exchanges. It adds simple and complex datatypes to XML documents and replaces DTD functionality with an XML Schema definition XML document.

XMLType

An XMLType column stores XML data using an underlying CLOB column in the database.

XMLType views

Oracle XML DB provides a way 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 need to migrate it to an XML format.

XPath

The open standard syntax for addressing elements within a document used by XSL and XPointer. XPath is currently a W3C recommendation. It specifies the data model and grammar for navigating an XML document utilized by XSLT, XLink and XML Query.

XPointer

The term and W3C recommendation to describe a reference to an XML document fragment. An XPointer can be used at the end of an XPath-formatted URI. It specifies the identification of individual entities or fragments within an XML document using XPath navigation.

XSL

See eXtensible Stylesheet Language.

XSLFO

See eXtensible Stylesheet Language Formatting Object.

XSLT

See eXtensible Stylesheet Language Transformation.

XSQL

The designation used by the Oracle Servlet providing the ability to produce dynamic XML documents from one or more SQL queries and optionally transform the document in the server using an XSL stylesheet.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X