Oracle*8i*®

Oracle*8i* Application Developer's Guide - XML

Release 3 (8.1.7)

September 2000

Part No. A86030-01

ORACLE®

Oracle8*i* Application Developer's Guide -XML, Release 3 (8.1.7)

Part No. A86030-01

Primary Author: Shelley Higgins

Graphics: Valerie Moore

Contributing Authors: Sandeepan Banerjee, Kishore Bhamidipati, Stefan Buchta, Dipto Chakravarty (Artesia Technologies, Inc.), Robert Dell'immagine, Brajesh Goyal, Robert Hall, Karun K, Stefan Kiritzo,Vishu Krishnamurthy, Murali Krishnaprasad, Olivier LeDiouris, Bryn Llewellen, Roger Medlin (Artesia Technologies, Inc.), Steve Muench,Visar Nimani, Paul Nock, Rajesh Raheja, Tomas Saulys, Mark Scardina, Flora Sun, Prabhu Thukkaram

Contributors: Ari Adler, Omar Alonso, Phil Bates, Mark Bauer, Ravinder Booreddy, Steve Cave, Steve Corbett, Claire Dessaux, Janet Lee, Shailendra Mishra, Andy Page, Rahul Pathak, Padmini Ranganathan, Den Raphaely, Jim Rawles, David Saslav, Chitra Sharma, Ena Singh, Keith Swartz, Kurt Thompson, Melanie Watson, Jon Wilkinson

Contents

2 Business Solutions Using Oracle XML

3 Oracle XML Components and General FAQs

6 Customizing Content with XML: Dynamic News Application

7 Personalizing Data Display With XML: Portal-to-Go

8 Customizing Presentation with XML and XSQL: Flight Finder

Part IV Data Exchange Using XML

9 Using Oracle Advanced Queuing (AQ) in XML Data Exchange

10 B2B: How *i*Procurement Uses XML to Offer Multiple Catalog Products to Users

11 Customizing Discoverer 3i Viewer with XSL

14 Using JDeveloper to Build Oracle XML Applications

15 Using Internet File System (iFS) to Build XML Applications

16 Building n-Tier Architectures for Media-Rich Management using XML: ArtesiaTech

Part VI XDK for Java

17 Using XML Parser for Java

18 Using XML Java Class Generator

19 Using XSQL Servlet

20 Using XML Transviewer Beans

Part VIII XDK for C++

22 Using XML Parser for C++

A An XML Primer

B Comparing Oracle XML Parsers and Class Generators by Language

C XDK for Java: Specifications and Cheat Sheets

D XDK for C: Specifications and Cheat Sheets

E XDK for C++: Specifications and Cheat Sheet

F XDK for PL/SQL: Specifications and Cheat Sheets

G XML-SQL Utility (XSU) Specifications and Cheat Sheets

Send Us Your Comments

Oracle8*i* Application Developer's Guide - XML, Release 3 (8.1.7)

Part No. A-86030-01

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- 7228 Attn: Server Technologies, Information Development
- Postal service:
  Oracle Corporation
  Server Technologies - Information development
  500 Oracle Parkway MS4op12
  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 introduces you to Oracle XML technology and presents several ways of implementing the technology. It includes case studies, numerous examples and sample applications. The case studies are presented according to their main function, namely, whether they are primarily used for either of the following:

- Content or Document Management
- Data Exchange and Business--to-Business purposes.

After introducing you to the main criteria to consider when designing your database-based application using the Oracle XML components, the manual then suggests some application scenarios and describes how to use the XML components together.

Examples and Sample Code

Many examples in the manual are provided with your software in the $ORACLE_HOME/xdk/java/demo/ or sample/ directory, or the $ORCLE_HOME/rdbms/demo directory.

One detailed application is described in Chapter 13, "B2B XML Application: Step by Step". This application illustrates how to implement an XML data exchange and customized presentation application.

Pre-Authored XML or Generated XML

XML documents are processed in one of two ways:

- Pre-Authored XML stored in LOBs
- Generated XML stored in relational tables with XML tags mapped to the respective columns in tables

XML Components

The Oracle XML components are available in four language implementations:

- Java, with the XDK for Java and XML-SQL Utility for Java
- PL/SQL, with the XDK for PL/SQL and XML-SQL Utility for PL/SQL
- C, with the XDK for C
- C++, with the XDK for C++

How to use each XML component is described in Chapter 4, "Using XML-SQL Utility (XSU)" and in Parts VI through IX.

Intended Audience

This guide is intended for developers building XML applications on Oracle8*i*.

Prerequisite Knowledge

An understanding of XML and XSL is helpful but not essential to use this manual. For your convenience, an XML primer is included in the appendix.

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

Related Manuals

Refer to the following manuals for more information:

- *Getting to Know Oracle8i* for information about the differences between Oracle8*i* and the Oracle8*i* Enterprise Edition and the available features and options. That book also describes all the features that are new in Oracle8*i*.
- *The JDeveloper Guide*
- *Oracle8i Application Developer's Guide - Fundamentals*
- *Oracle8i Application Developer's Guide - Advanced Queuing*
- *Oracle Integration Server Overview*
- *Oracle8i XML Reference*

Feature Coverage and Availability

Information in this manual represents a snapshot of information on Oracle XML technology components. These are changing rapidly. In order to view the latest information, refer to Oracle Technology Network (OTN) site:
http://technet.oracle.com/tech/xml

How this Book is Organized

The book is organized into 9 parts, 24 chapters, and 7 appendixes. It includes an index and glossary.

Part I Introducing Oracle XML

- Chapter 1, "Introduction to Oracle XML", introduces you to the Oracle XML components, tools used to build XML applications, interMedia Text. It also describes and the issues for building XML applications on Oracle8*i*. This chapter includes a 'roadmap' to the information presented in this manual.
- Chapter 2, "Business Solutions Using Oracle XML", briefly describes how Oracle XML components can be used in typical content / document management and business-to-business messaging applications.
- Chapter 3, "Oracle XML Components and General FAQs", introduces you to the Oracle XML components, the XML Development Kits and XML-SQL Utility. It also summarizes the different ways you can generate XML documents for each language, Java, C, C++, and PL / SQL. It also provides Frequently Asked Questions (FAQs) that include a variety of general questions about Oracle XML.

Part II XML-SQL Utility (XSU)

- Chapter 4, "Using XML-SQL Utility (XSU)", describes how to use XML-SQL Utility Java and PL / SQL versions to generate and 'store' XML documents, how to INSERT / UPDATE / DELETE XML documents in the database, use the command line tool, map elements to columns. Examples in this chapter are available from $ORACLE_HOME / rdbms / demo / xsu. This chapter also provides Frequently Asked Questions (FAQs).

Part III Managing Content and Document with XML

- Chapter 5, "Using interMedia Text to Search and Retrieve Data from XML Documents", introduces you to interMedia Text, using the CONTAINS operator, how to create an interMedia Text index, how to build a query, and text query expressions. It also describes some basics about using the PL / SQL supplied package, CTX_DDL and the XML_SECTION_GROUP and its attributes, and the AUTO_SECTION_GROUP. This chapter also provides Frequently Asked Questions (FAQs).
- Chapter 6, "Customizing Content with XML: Dynamic News Application", describes the Dynamic News application, the three servlets used in the application, how XML-SQL Utility is used to access news data from Oracle8i, the three levels of user customization — static, semi-dynamic, and dynamic. This chapter also includes details about customizing data presentation.
- Chapter 7, "Personalizing Data Display With XML: Portal-to-Go", describes the portal-to-go components and how they are used to extract content from a

browser web site, convet this to XML, and transform this for display on a variety of devices.

- Chapter 8, "Customizing Presentation with XML and XSQL: Flight Finder", describes how Flight Finder generates XML to and from the database and uses XSQL Servlet to process queries and output the results as XML. It also discusses how Flight Finder formats XML data using stylesheets. This demo and application is available on Oracle Technology Network (OTN).

Part IV Data Exchange Using XML

- Chapter 9, "Using Oracle Advanced Queuing (AQ) in XML Data Exchange", introduces you to some Advanced Queueing concepts and describes how AQ and XML complement each other. It includes one Java AQ example and one PL/SQL AQ example. This chapter also provides several FAQs.
- Chapter 10, "B2B: How iProcurement Uses XML to Offer Multiple Catalog Products to Users", describes the main components of *i*Procurement and how *i*Procurement uses XML Parser for Java to parser and check incoming 3rd party catalogs and generally provide catalog content management. The DTD used by iProcurement is described in detail. The "unified" catalog is extracted form the database, loaded, and sent to Oracle Applications using a PL/SQL program.
- Chapter 11, "Customizing Discoverer 3i Viewer with XSL", describes how Discoverer 3i Viewer is used to customize web applications and customize presentation using stylehseets. It includes several FAQs.
- Chapter 12, "Phone Number Portability Using XML Messaging", introduces you the Phone Number Portability application and summarizes how XML messaging is used in *i*Message Studio, Event Manager, and Adapters.

Part V Developing Applications Using Oracle XML

- Chapter 13, "B2B XML Application: Step by Step", describes in detail how to build and implement a B2B XML application using XSQL servlet and transform the XML message according to different user devices. This application also uses simple AQ messaging.
- Chapter 14, "Using JDeveloper to Build Oracle XML Applications", introduces you using JDeveloper for building XML applications, using XSQL servlet from JDeveloper, and steps to take when about building a Mobile application with JDeveloper. This chapter includes FAQs.
- Chapter 15, "Using Internet File System (iFS) to Build XML Applications", introduces you to Internet File System (*i*FS) and its XML features.

- Chapter 16, "Building n-Tier Architectures for Media-Rich Management using XML: ArtesiaTech", describes an advanced multi-tier XML messaging architecture for managing digital assets, such as video clips. It discusses object-oriented messaging with XML and compares XML and IDL.

Part VI XDK for Java

- Chapter 17, "Using XML Parser for Java", describes ways of using XML Parser for Java and XSLT Processor. It lists the examples provided with the software. This chapter includes FAQs.
- Chapter 18, "Using XML Java Class Generator", describes ways of using XML Java C;ass Generator. It lists the examples provided with the software. This chapter includes FAQs.
- Chapter 19, "Using XSQL Servlet", provides some insite on using XSQL Servlet. It includes diagrams that explain how the XSQL Page Processor works.This chapter includes FAQs.
- Chapter 20, "Using XML Transviewer Beans", discusses the XML Transviewer Beans and how to use them. It lists the examples provided with the software.

Part VII XDK for C

- Chapter 21, "Using XML Parser for C", describes ways of using XML Parser for C and XSLT Processor. It lists the examples provided with the software.

Part VIII XDK for C++

- Chapter 22, "Using XML Parser for C++", describes ways of using XML Parser for C++ and XSLT Processor. It lists the examples provided with the software.
- Chapter 23, "Using XML C++ Class Generator", describes ways of using XML C++ Class Generator. It lists the examples provided with the software.

Part IX XDK for PL/SQL

- Chapter 24, "Using XML Parser for PL/SQL", describes ways of using XML Parser for PL/SQL and XSLT Processor. It lists the examples provided with the software. This chapter includes FAQs.

Appendix A, "An XML Primer", introduces you to some basic and background information about XML.

Appendix B, "Comparing Oracle XML Parsers and Class Generators by Language", compares the Oracle XML Parsers and Class Generators according to implementation language.

Appendix C, "XDK for Java: Specifications and Cheat Sheets", describes the XDK for Java component specifications. Includes several top level class and method listings.

Appendix D, "XDK for C: Specifications and Cheat Sheets", describes the XDK for C specifications. Includes top level function listings.

Appendix E, "XDK for C++: Specifications and Cheat Sheet", describes the XDK for C++ component specifications. Includes several top level class and method listings.

Appendix F, "XDK for PL/SQL: Specifications and Cheat Sheets", describes the XDK for PL/SQL specifications. Includes several top level function listings.

Appendix G, "XML-SQL Utility (XSU) Specifications and Cheat Sheets", describes the XML-SQL Utility (XSU) for Java and PL/SQL specifications. Includes several top level method and function listings.

Conventions Used in this Guide

This section explains the conventions used in this book:

Text

The text in this reference adheres to the following conventions:

UPPERCASE	Uppercase text calls attention to SQL keywords, filenames, and initialization parameters.
italics	Italicized text calls attention to parameters of SQL statements.
boldface	Boldface text calls attention to definitions of terms.
`names of methods, functions, executables, ...`	monospaced font text callas attention to names of methods, functions, and other executables.

Acronym List

The following table lists acronyms used thoughout this manual. For further explanation of these terms refer to the "Glossary".

Acronym	Description
API	Application Program Interface

Acronym	Description
B2B	Business-to-Business applications
B2C	Business-to-Consumer applications
BC4J	Business Components for Java
CDF	Channel Definition Format. Provides a way to exchange information about channels on the internet.
CSS	Cascading Style Sheets
DDT	Document Type Definition
DOM	Document Object Model
EDI.	Electronic Data Interchange
HTML	Hypertext Markup Language
OAG	Open Applications Group
OAI	Oracle Applications Integrator
OE	Oracle Exchange.
PDA	Personal Digital Assistant
RDBMS	Relational Database Management System
RDF	Resource Definition Framework
SAX	Simple API for XML
SGML	Standard Generalized Mark Up Language
W3C	World Wide Web Consortium
WBEM	Web-Based Enterprise Management
XDK	Oracle XML Development Kit
XLink	XML Linking Language
XML	eXtensible Markup Language
XMLQuery	W3C effort to specify a query language for XML documents
XML Schema	W3C specification. XML Schema Processor automatically ensures validity of XML documents and data.
XPath	W3C recommendation specificying data model and grammar for navigating an XML document

Acronym	Description
XPointer	W3C recomendation that specifies the identification of individual entities or fragments within an XML document
XSL	(W3C) eXtensible Stylesheet Language
XSL Stylesheet	Specifies the presentation of a class of XML documents by describing how an instance of the class is transformed
XSL-T	XSL Transformation

How to Order this Manual

To order this manual, carry out these steps:

- If you are not an Oracle employee, go to the following site:

  http://store.oracle.com

  At the top of the home page, select Database > Documentation. Under Oracle8i Documentation, select Release 8.1.7 and scroll down to find this manual, "Application Developer's Guide-XML". Enter the quantity you need. Select "Buy". Follow the instructions.

- If you are an Oracle employee, to order this manual, go to the following site:

  http://store-inside.us.oracle.com

  At the top of the home page, select Database > Documentation. Under Oracle8i Documentation, select Release 8.1.7 and scroll down to find this manual, "Application Developer's Guide-XML". Enter the quantity you need. Select "Buy". Follow the instructions.

Part I

Introducing Oracle XML

Part I of the book introduces you to XML, Oracle XML and features, where you can use the Oracle XML components, demonstrations and example scenarios, and the Oracle XML components.

It contains the following chapters:

- Chapter 1, "Introduction to Oracle XML"
- Chapter 2, "Business Solutions Using Oracle XML"
- Chapter 3, "Oracle XML Components and General FAQs"

1

Introduction to Oracle XML

This chapter describes the following sections:

- Oracle XML Samples and Demos
- What's Needed to Run Oracle XML Components
- XML Technical Support is Free on OTN

Introduction

This manual describes Oracle*8i*'s XML-enabled database technology, how XML data can be stored, managed, and queried in the Oracle*8i* database, and how to use Oracle*8i* XML technology and the appropriate Oracle development tools to build applications.

Oracle XML in Action: Applications

Authored XML or Generated XML

This manual focuses on describing information that will assist you in building *database* XML applications using Oracle XML technology. Be aware, that in general, your application will be processing XML documents that are in one of two overall formats:

- *Authored XML Documents*: These are XML documents stored as XML documents, in CLOBs, in the database.
- *Generated XML Documents*: These are XML documents that are stored as *relational data* in tables or views the database. This data can then be generated back into XML format, dynamically, when necessary.

Oracle XML Applications

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

Content and Document Management

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

See: Content and Document Management Chapters:

- Chapter 6, "Customizing Content with XML: Dynamic News Application"
- Chapter 7, "Personalizing Data Display With XML: Portal-to-Go"
- Chapter 8, "Customizing Presentation with XML and XSQL: Flight Finder"

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

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

See: B2B Chapters:

Chapter 9, "Using Oracle Advanced Queuing (AQ) in XML Data Exchange"

Chapter 10, "B2B: How iProcurement Uses XML to Offer Multiple Catalog Products to Users"

Chapter 11, "Customizing Discoverer 3i Viewer with XSL"

Chapter 12, "Phone Number Portability Using XML Messaging"

Chapter 13, "B2B XML Application: Step by Step"

Roadmap of this Manual

Figure 1–1, "Oracle XML Components and e-Business Solutions: Roadmap"lists a few XML-based business solutions for content and document management as well as B2B and B2C messaging.

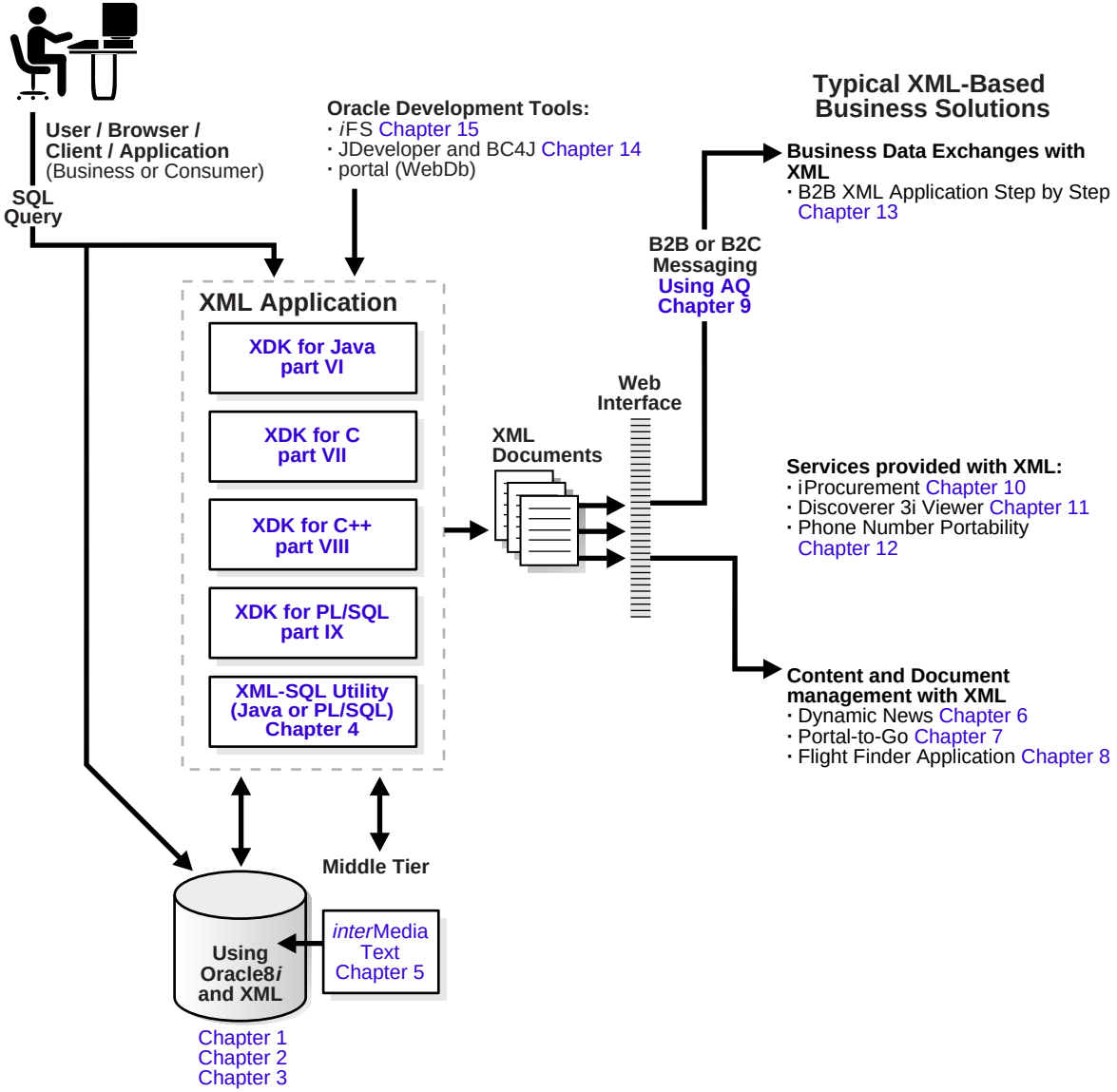
The figure depicts an SQL query being placed by a user from a browser or application. The query being processed by a 'black box', called "XML Application".The XML application accesses Oracle8i. The application then produces XML documents which may or may not be passed via a web interface as part of a data exchange solution, to provide services, or for content and document management and presentation versatility to other businesses or applications within the same business.

Besides introducing you to the main components involved in Oracle XML, Figure 1–1 also serves as a roadmap for this manual. The figure maps all the main "stops" in the manual. In the online versions of this manual, if you click on these "stops" you will go directly to the chapters of interest.

- Beginning with the introductory and basic information about using Oracle8i and where to use the Oracle XML components (Chapters 1,2, and 3), and how to apply *inter*Media Text to search and retrieve information from XML documents.
- Typical XML-based business solutions, involving content management and presentation are covered in Chapters 6, 7,and 8.
- Typical XML-based business solutions, involving B2B and data exchange are covered in Chapters 9, 10, 11, and 12.
- Some introductory information about using JDeveloper and iFS, as well as a detailed step by step code intensive application implementation are covered in Chapters 13, 14, and 15.
- Last but first in importance, the roadmap shows the "how to use" the various Oracle XML components, in the "XML Application" box. The "how to use" information is covered in Chapter 4 and Chapters 16 through 23. Not included in the roadmap are the Appendixes which are really just more information about Chapters 4, 16 through 23.

The one chapter not shown in this roadmap is the chapter discussing strategy for building n-tier XML applications, Chapter 16, "Building n-Tier Architectures for Media-Rich Management using XML: ArtesiaTech".

Figure 1–1 Oracle XML Components and e-Business Solutions: Roadmap



Oracle XML

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

What is Oracle XML?

XML models structured and semi-structured data and is now the accepted primary model for internet data.

Oracle8i has evolved to *support* complex, structured, unstructured, and semi-structured data. It is also XML-enabled and facilitates the storage, query, presentation, and manipulation of XML data.

For more information on how Oracle*8i* XML supports structured and semi-structured data, read on.

Chapter 4, "Using XML-SQL Utility (XSU)" will cover in detail how to generate XML documents from a database and how to store the documents back into the database. Further techniques are discussed in Chapters 16 through 23. In particular, Chapter 19, "Using XSQL Servlet" gives many practical guidelines for using Oracle XML technology and components with Oracle8*i*.

Oracle XML Components

Figure 1–2 shows the Oracle XML components in the box "XML Application".

Oracle XML components are comprised of the following:

- XDK[1] for Java
  - XML Parser for Java and XSL-T Processor
  - XML Java Class Generator
  - XSQL Servlet
  - XML Transviewer Beans
  - XML Schema Processor[2]
- XDK for C

[1] XDK - XML Developer's Kit

    - XML Parser for C
- XDK for C++
    - XML Parser for C++
    - XML C++ Class Generator
- XDK for PL/SQL
    - XML Parser for PL/SQL
- XML-SQL Utility (XSU) for Java and PL/SQL

Figure 1–2 also lists the following typical XML-based business solutions:

- *Business Data Exchanges with XML*
    - Buyer-Supplier Transparent Trading Automation
    - Seamless integration of partners and HTTP-based data exchange
    - Database inventory access and integration of commercial transactions and flow
    - Services provided with XML. These also fall under
        * Self-service procurement, such as using Oracle *i*Procurement
        * Data mining and reporting with Oracle Discoverer 3i Viewer
        * Oracle Exchange and Applications
        * Phone number portability

    These data exchanges use Oracle Advanced Queuing (AQ).
- *Content and Document Management with XML*
    - Personalized publishing and portals
    - Customized presentation. Dynamic News case study, Portal-to-Go, and Flight Finder

[2] The Oracle XML Schema processor will be described in future versions of this manual. You can currently read about it and download it from http://technet.oracle.com/tech/xml. C, C++, and PL/SQL versions are forthcoming.

Figure 1–2 Oracle XML Components and e-Business Solutions: What's Involved

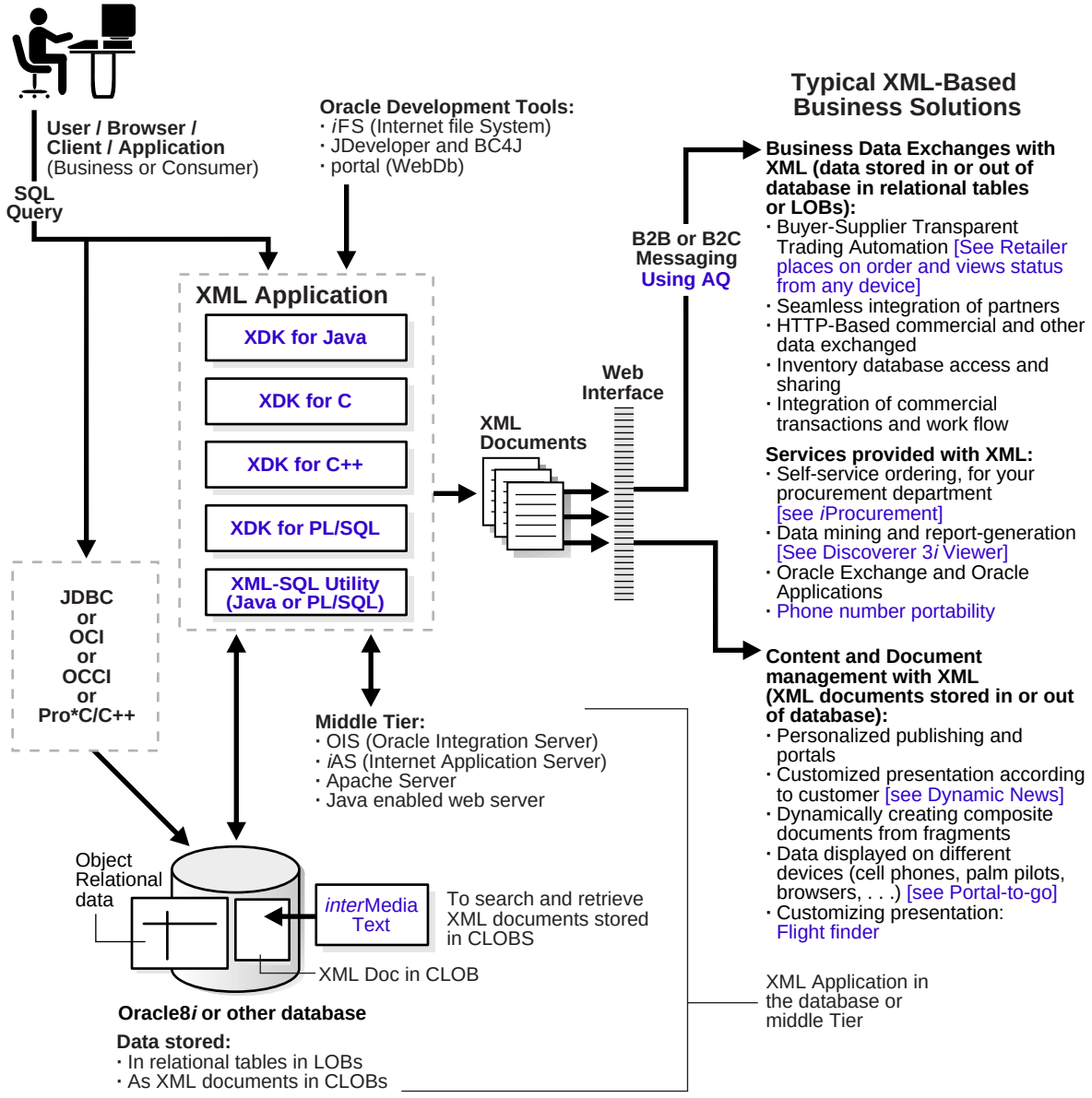


Figure 1–2 also shows the following:

Oracle Development Tools

Internet File System (*i*FS), JDeveloper, and Portal (WebDB) can be used to build XML applications. These tools are summarized here "The Oracle Suite of Integrated Tools" on page 1-13.

Database and Middle Tier

The XML application can either reside on the database or a middle tier, such as OIS, *i*AS, Apache Server, or other Java enabled web server.

Data Stored in the Database

Data is stored as relational tables or as XML documents in CLOBs. *inter*Media Text used to search and retrieve from XML document stored in CLOBs.

When to Use Oracle XML Components: How They Work Together

For descriptions of the Oracle XML components and how they work together see Chapter 2, "Business Solutions Using Oracle XML" and Chapter 3, "Oracle XML Components and General FAQs".

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

Oracle8*i* XML Features

Oracle8*i* XML features include the following:

- Based on XML Open Standards
- Deployment and Application
- Supports Multiple Languages and Platforms
- Scalability and Performance
- Supports applications and servers

Based on XML Open Standards

Oracle XML components are based on the following W3C XML recommendations:

- XML 1.0 Core

- DOM 1.0
- XSLT 1.0
- XML Namespaces 1.0
- XML Schema. This component is available for download from http://technet/tech/xml however it is not described in this version of the manual. It will be described in a forthcoming release.
- XPath. This recommendation has not been made publicly available with this release of Oracle XML.

For further information on W3C activities, see http://www.w3c.org

Deployment and Application

Oracle XML technology provides a versatile infrastructure for use in flexible deployment architectures. It uses any combination of the following:

- Relational databases
- Middle tier systems:
  - Web servers, such as iAS (Internet Application Server)
  - Integration servers, such as Oracle Integration Server — OIS
- Thin or thick clients

Supports Multiple Languages and Platforms

Oracle XML supports the following:

- Platform independent languages:
  - Java and Java2
  - PL/SQL
- Platform dependent languages:
  - C++
  - C
- Platforms
  - All Oracle*8i* platforms, including Solaris, Win32, Linux, and HP-UX

Scalability and Performance

Oracle8*i* XML supports the following scalability and performance related features:

- On-demand presentation through the application of stylesheets at runtime.
- Client-server performance through the use of server-side processing.
- Intelligent caching. Individual XML components cache stylesheets, DTDs, and documents, and work seamlessly with middle-tier caches such as *i*Cache.
- Integrated load-balancing. The XML components are architected in a tiered approach to reduce bottle-necks and handle dynamic loading.
- Connection pooling. The XML components have integrated support for maintaining a pool of connections that enhance scalability.

Why Use Oracle*8i* XML?

Oracle*8i* is well-suited for building XML database applications for the following reasons:

- The Oracle Suite of Integrated Tools
- Indexing and Searching XML Documents with Oracle interMedia Text
- Messaging Hubs and Middle Tier Components
- Back-End to Database to Front-End Integration Issues
- Oracle XML Parser Provides the Two Most Common APIs: DOM and SAX
- Loading XML into a Database
- Oracle XML Samples and Demos

The Oracle Suite of Integrated Tools

Oracle*8i* provides an integrated suite of tools for building e-business applications:

- Oracle JDeveloper 3.2 and Oracle Business Objects for Java (BC4J)
- Internet File System (iFS)
- Portal (webDB)
- Oracle Exchange
- XML Gateway

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

Oracle JDeveloper 3.2 and Oracle Business Objects for Java (BC4J)

Oracle JDeveloper 3.2 is an integrated environment for building, deploying, and debugging applications leveraging Java and XML on Oracle*8i*. It facilitates working in Java 1.1 or 1.2 with CORBA, EJB, and Java Stored Procedures. With it you can do the following:

- Directly access Oracle XML components to build multi-tier applications
- Create and debug Java Servlets that serve XML information in a snap
- Build portable application logic with JDeveloper and BC4J components

Examples of applications built using Oracle JDeveloper include:

- *i*Procurement (Self Service Applications) including Self-Service Web-Expensing. See Chapter 10, "B2B: How iProcurement Uses XML to Offer Multiple Catalog Products to Users".
- Content Delivery for PDAs. See Chapter 7, "Personalizing Data Display With XML: Portal-to-Go".
- Online Marketplaces
- Retailer - Supplier transaction using XML and AQ messaging. See Chapter 13, "B2B XML Application: Step by Step".

See Chapter 14, "Using JDeveloper to Build Oracle XML Applications" for more information on JDeveloper and XML applications.

Oracle Business Components for Java (BC4J) Business Components for Java (BC4J) is an Oracle8*i* application framework for encapsulating business logic into reusable libraries of Java components and reusing the business logic through flexible, SQL-based views of information.

Internet File System (*i*FS)

Access to Oracle8*i* Internet File System (*i*FS) facilitates organizing and accessing documents and data using a file- and folder-based metaphor through standard Windows & Internet protocols such as SMB, HTTP, FTP, SMTP, and IMAP4.

*i*Fs facilitates building and administering web-based applications. It is an application interface for Java and can load a document, such as a Powerpoint .PPT file, into Oracle8*i* (O817) and display the document from a web server, such as *i*AS or Apache Web Server. See also Chapter 15, "Using Internet File System (iFS) to Build XML Applications".

*i*FS is a simple way for developer's to work with XML, where *i*FS serves as the repository for XML. *i*FS automatically parses XML and stores content in tables and columns. *i*FS renders the content when a file is requested delivering select information, for example, on the web.

For more information see http://technet.oracle.com/products/ifs/

Portal (webDB)

Portal can for example input XML-based Rich Site Summary (RSS) format documents, then merge the information with an XSL stylesheet so that it can be

rendered in a browser. This design efficiently separates the rendition of information from the information itself and allows for easy customization of the look-and-feel without risk to data integrity.

Portal is part of Oracle Portal: Oracle Portal is software for building and deploying enterprise portals, the Web sites that power an e-business. The browser interface delivers an organized, personalized view of business information, web content, and applications needed by each user. It includes site-building and self-service Web publishing functionality of WebDB 2.2 and adds new enterprise portal features such as single sign-on, personalization, and content classification. Oracle Portal uses Oracle8*i* and is deployed on Oracle *i*AS. It is packaged with *i*AS.

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

Oracle Exchange

The Oracle Exchange platform is based on Oracle8*i*. It offers all necessary business transactions to support an entire industry's or a company's supply chain. Oracle Exchange is based on Oracle's e-Business Suite, which supports a supply chain from the initial contact with the prospect, to manufacturing planning and execution, to post sales on-going service and support.

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

Other Initiatives

Besides these tools, the following initiatives are underway:

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

Support for XML Metadata Interchange (XMI) specification proposed by Oracle, IBM, and Unisys. This enables application development tools and data warehousing tools from Oracle and others to exchange common metadata, insuring

that you can choose any tool without having to modify your application and warehouse design.

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

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

Indexing and Searching XML Documents with Oracle *inter*Media Text

*inter*Media Text provides powerful search and retrieval options for XML stored in CLOBs and other documents. It can index and search XML documents and document 'sections' of any size up to 4 Gigabytes each stored in a column in a table.

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

XML documents or document sections saved into "text CLOBs" in the database can be enabled for indexing by Oracle*8i inter*Media Text's text-search engine. Developer's can pinpoint searches to data within a specific XML hierarchy as well as locate name-value pairs in attributes of XML elements.

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

> **See Also:** Chapter 5, "Using interMedia Text to Search and Retrieve Data from XML Documents" and *Oracle8i interMedia Text Reference*

Messaging Hubs and Middle Tier Components

Also included in the Oracle XML are the following components:

- *XML-Enabled Messaging Hubs*, such as Oracle XML Gateway. These hubs are vital in business-to-business applications that interface with non-Oracle systems. See also Chapter 9, "Using Oracle Advanced Queuing (AQ) in XML Data Exchange".

- *Middle Tier Systems*: XML-enabled application, web, or integrated servers, such as Oracle Integration Server (OIS) and Internet Application Server (*i*AS).

Oracle*8i* JVM (Java Virtual Machine)

Built from the ground up on Oracle Multi-threaded Server (MTS) architecture, Oracle8*i* JVM (Jserver) is a Java 1.2 compliant virtual machine that data server shares memory address space. This allows the following:

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

Oracle*8i* JVM supports native CORBA and EJB standards as well as Java Stored Procedures for easy integration of Java with SQL and PL/SQL.

Oracle Integration Server (OIS)

Oracle Integration Server sends and receives XML payload messages using Oracle Advanced Queueing. Oracle Message Broker packages and delivers the XML messages using JMS wrappers.

Oracle Internet Application Server (iAS)

Oracle Internet Application Server 8i (Oracle *i*AS), offers services for both intranet and Internet web applications. It is integrated with Oracle8i and offers advanced services like data caching and Oracle Portal.

Back-End to Database to Front-End Integration Issues

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

Such data exchange between different vendors' relational and object-relational databases is simpler using XML. One example of a data exchange implementation using XML and AQ is provided in Chapter 13, "B2B XML Application: Step by Step".

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

Higher Performance Implications

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

- Processing database data and XML together on the same server, helps eliminate network traffic for data access.
- Exploiting the speed of the Oracle8i's query engine and Oracle8i JVM, iAS, or OIS further enhances data access speed.

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

Oracle XML Parser Provides the Two Most Common APIs: DOM and SAX

Oracle XML Parser is implemented in four languages, Java, C, C++, and PL/SQL. The Java version runs directly on Oracle*8i* JVM (Java virtual machine). It supports the XML 1.0 specification and is used as a validating or non-validating parser.

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

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

For more information, see Chapter 17, "Using XML Parser for Java"". See Appendix B, "Comparing Oracle XML Parsers and Class Generators by Language", for a comparison of the Oracle XML parsers and generators.

Writing Custom XML Applications

Hence writing custom applications that process XML documents can be simpler in an Oracle*8i* environment. This enables you to write portable standards-based applications and components in your language of choice that can be deployed on any tier.

The XML parser is part of the Oracle*8i* platform on every operating system where Oracle*8i* is ported.

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

Loading XML into a Database

You can to use the following options to load XML data or DTD files into Oracle8*i*:

- Use PL/SQL stored procedures for LOB, such as `dbms_lob`
- Write Java (Pro*C, C++) custom code
- Use sql*loader
- Use Oracle intermedia
- XML-SQL Utility

You can also use Internet File System (*i*FS) to put an XML document into the database. However, it does not support DTDs. It will however be supporting XML Schema which is the standard that will replace DTD.

Storing and Extracting XML Data from Oracle8*i*

XML has emerged as the standard for data interchange on the web. Oracle8*i* is XML-enabled to handle the current needs of the market.

Oracle8*i* is capable of storing the following:

- Structured XML data as object-relational data
- Unstructured XML document as *inter*Media Text data

Oracle8*i* provides the ability to automatically extract object-relational data as XML. Efficient querying of XML data is facilitated using standard SQL.

It also provides the ability to access XML documents using the DOM (Document Object Model) API.

Oracle*8i*: Object-Relational Infrastructure

Oracle has evolved its database to an object-relational engine following the SQL:1999 standard. Oracle's object-relational engine allows you to define *object* types, *collections* of types, and *references* to object types.

This object-relational infrastructure provides the support for storing structured object instances in the database. XML, inherently being a structured data format, can be easily mapped to object-relational instances. This mapping will be discussed in "XML Schemas and Mapping of Documents".

Oracle*8i*: Extensible Architecture

Normally, the database provides a set of services - for example, a basic storage service, a query processing service, services for indexing, query optimization, and so on.

Extensible Services

In Oracle8*i*, these services are made *extensible* so that data cartridges can provide their own implementations. When some aspect of a native service provided by the database is not adequate for a specific domain, a developer can build a domain-specific implementation.

For example, if you build a Spatial data cartridge for Geographical Information Systems, you may need the capability to create spatial indexes. To do this, you would implement routines that create a spatial index, insert an entry into the index, update the index, delete from it, and so on. The server would then automatically

invoke your implementation every time indexing functionality was needed for spatial data. In effect, you would have extended the Indexing Service of the server to handle spatial data.

Extensibility and XML

Extensibility enables special indexing on XML (including Text indexes for section searching), special operators to process XML, aggregation of XML, and special optimization of queries involving XML.

*inter*Media Text Searching

An example of the extensibility infrastructure is *inter*Media Text searching. The text kept in LOBs is indexed using the extensibility indexing interface. *inter*Media text provides operators such as CONTAINS which you can use to search within the text for substring matches.

The Oracle*8i* extensibility framework provides the infrastructure for specialized XML cartridges to be built, where indexing and optimization access to XML is accomplished by the cartridge.

Oracle8i Supports Native Java

Oracle8*i* provides native support for Java in the DBMS, by providing a native Java VM that is closely integrated with the database for high performance and scalability. In addition, the database system natively supports JDBC, SQLJ, an ORB and the EJB framework. In addition, Oracle8*i* also comes with a HTTP listener, which means that it can act as a web server as well.

The object-relational framework provides a more natural way to maintain a consistent structure between a set of Java classes at the application level and the data model at the data storage level. In Oracle8*i*, the object-relational facilities have been tightly integrated with the Java environment in the following ways:

- Server object-relational schema can be mapped to java classes. The JPublisher utility can generate this mapping automatically.
- Java is one of the language choices for implementing object type methods and data cartridges.
- Objects can be manipulated (stored and retrieved) using JDBC or SQLJ.

Support for Java within the database is vital, since a lot of the XML infrastructure, such as parsers etc. are available in Java and can be readily used inside the server.

Also, the components for XML built in Java can be run inside the server or outside in the application tier.

XML in the Database: Generated and Authored XML

Oracle8*i* supports different aspects to using XML in the database.

- *"Generated XML"*: Using XML as an Interchange Format. The most common way of using XML in the database is to use XML as a interchange format where existing business data is wrapped in XML structures. In this case the XML format is used only for the interchange process itself and is transient.
- *"Authored XML"*: Store and query XML documents in the database.

Generated XML

XML can be generated from the data stored in your database tables and views using the XML-SQL Utility (XSU). The XML-SQL Utility consits of a command line front end, a Java API, and a PL/SQL API. The XML-SQL Utility is also available for download from the Oracle Technology Network (OTN).

XML-SQL Utility (XSU) Converts SQL Query Results into XML

This utility converts the result of an SQL query into XML by mapping the query alias' or column names into the element tag names and preserving the nesting of object types. The XML-SQL Utility can return the XML in a string or DOM tree (Document Object Model) representation, where the latter is a format very convenient for further manipulation.

Mapping Between XML and the Database

There is a clean relationship between structured XML instances and object-relational types:

- Columns map to top level elements.
- Scalar values map to a elements with text only content.
- Object types are mapped to elements, and the attributes of the object type make up sub-elements. Collections map to lists of elements.

Example: Using XML-SQL Utility to Get the XML Representation of the Result of an SQL Query

The following example uses XML-SQL Utility's Java API. In the example we specify a SQL Query which is then applied against the database. The result of the query is returned as XML.

```
public void testXML()
 {
   DriverManager.registerDriver(
      new oracle.jdbc.driver.OracleDriver());

   //initialize a JDBC connection
   Connection conn =
        DriverManager.getConnection(
                    "jdbc:oracle:oci8:scott/tiger@");

   //initialize the OracleXMLQuery;
   OracleXMLQuery qry =
         new OracleXMLQuery(conn,
                   ”select * from purchaseOrderTab”);

   // set the rowset element name
   qry.setRowsetTag(“PurchaseOrderList”);
   // set the row element name
   qry.setRowTag(“PurchaseOrder”);
   // get the XML result
   String xmlString = qry.getXMLString();
    // print result
   System.out.println(“ OUPUT IS:\n"+xmlString);
}
```

Running the above code results in the following XML document:

```
<?xml version='1.0'?>
<PurchaseOrderList>
   <PurchaseOrder num=”1”>
     <purchaseNo>1001</purchaseNo>
     <purchaseDate>10-Jan-1999</purchaseDate>
     <customer>
        <custNo>100</custNo>
        <custName>Hose</custName>
        <custAddr>
           <street>200 Redwood Shrs</street>
           <city>Redwood City</city>
           <state>CA</state>
           <zip>94065</zip>
        </custAddr>
     </customer>
     <lineItemList>
        <lineItem>
           <lineItemNo>901</lineItemNo>
```

```
            <lineItemName>Chair</lineItemName>
            <lineItemPrice>234.55</lineItemPrice>
            <lineItemQuan>10</lineItemQuan>
         </lineItem>
         <lineItem>
            <lineItemNo>991</lineItemNo>
            <lineItemName>Desk</lineItemName>
            <lineItemPrice>3456.63</lineItemPrice>
            <lineItemQuan>20</lineItemQuan>
         </lineItem>
      </lineItemList>
   </PurchaseOrder>
   <PurchaseOrder>
      <!-- more purchase orders. -->
   </PurchaseOrder>
</PurchaseOrderList>
```

The XML document created is an exact structural replica of the queried database object(s). You can also use object-relational views to get structured XML (like above) from data stored in a flat relational schema, without using XSLT.

XML Storage Options

There are several options available for storing XML data:

- LOB Storage: Stores "Authored XML" Documents
- Object-Relational Storage: Stores "Generated XML" Documents

LOB Storage: Stores "Authored XML" Documents

Oracle8*i* supports the storage of 'large objects' or LOBs as 'character LOBs' (CLOB), 'binary LOBs' (BLOB), or externally stored 'binary files' (BFILE). LOBs are used to store "Authored or Native XML" documents.

Use CLOBs or BFILEs to Store Unstructured Data

CLOBs, which can store large character data, can be useful for storing unstructured XML documents. Although more useful for multi-media data, BFILEs which are external file references can also be used. In this case the XML is stored and managed outside the RDBMS, but can be used in queries on the server. The metadata for the document can be stored in object-relational tables in the server for fast indexing and access.

*inter*Media Text Indexing Supports Searching Content in XML Elements

Oracle8*i* allows the creation of *inter*Media text indexes on these LOB columns, in addition to URLs that point to external documents. This text cartridge leverages the extensibility mechanism and provides full text indexing of these documents. Oracle8*i* has extended this mechanism to work on XML data as well.

The text cartridge can recognize XML tags, and section and sub-section text searching have been extended to support searching within an XML element content. The result is that queries can be posed on unstructured data and restricted to certain sections or elements within a document.

Object-Relational Storage: Stores "Generated XML" Documents

A natural way to store XML is as object-relational instances. The object-relational type system can fully capture and express the nesting and list semantics of XML. Complex XML documents can be stored as object-relational instances and indexed efficiently. With the extensibility infrastructure, new types of indices, such as path indices, can be created for faster searching through XML documents.

Storage of Structured XML Documents ("Generated XML")

Data may be in the form of structured XML documents, where the structure is well defined and is the same for all instances. In this case, the document can be stored in relational or object-relational structures. In this case as well, the object-relational type system can provide a direct mapping to the XML document.

This mapping is relatively straight forward and Oracle XML-SQL Utility (XSU) offers an insert mechanism that can map an XML document directly into a given table or view.

XML-SQL Utility Stores XML Data By Preserving XML Structure

To reiterate, XML-SQL Utility stores XML data by preserving XML structure as follows:

- Tag names are mapped to columns
- Text-only data (elements) are mapped to scalar columns
- Elements which have sub-elements are mapped to object types
- Element with lists of sub-elements are mapped to collections...

Example: Using XML-SQL Utilitiy's JAVA API to Insert XML into the Database

The program below illustrates a simple use of XSU's JAVA API with the goal of inserting an XML document into a database table:

```
public void testInsertXML()
{
   DriverManager.registerDriver(
      new oracle.jdbc.driver.OracleDriver());

   //initialize a JDBC connection
   Connection conn =
        DriverManager.getConnection("jdbc:oracle:oci8:scott/tiger@");

   //create a URL object for the file/url containing the xml
   URL url = sav.getURL("sample.xml");
   OracleXMLSave sav = new OracleXMLSave(conn,"purchaseOrderTab");
   System.out.println("Rows inserted:  "+sav.insertXML(xmlDoc));
}
```

XML Document Object-Relational Storage: Advantages

The advantage of storing an XML document as an object-relational instance is that the structure of the document is preserved in the database as well. This allows the XML document to be viewed and traversed in SQL in a way similar to an XPath traversal on the document.

For instance an XPath traversal such as,

```
//purchase_order[pono=101]/shipaddr/street
```

can be easily represented as an attribute traversal in SQL:

```
SELECT   po.shipaddr.street
  FROM   purchase_order_tab po
  WHERE  po.pono = 100;
```

Mapping to Object-Relational storage enables existing database applications to work against XML data. Also, functionality provided by Oracle8*i* on Object-Relational columns, such as indexing, partitioning, and parallel query, can be leveraged.

XML Document Object-Relational Storage: Disadvantages

However, using such a mapping, the original document is not exactly reproducible. For instance, comments are lost. But this can be avoided by storing a copy of the original document in a CLOB as discussed in the following section, and using the object-relational mapped data for query efficiency purposes.

Another potential problem could arise due to the ordering amongst the elements. In order to preserve the element ordering, you can have a special column in the underlying table and order the results using that column.

Storage of Unstructured XML Documents ("Authored XML")

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

Oracle8*i* provides *inter*Media Text for indexing CLOB columns. This uses the extensibility mechanism to implement operators such as CONTAINS to search the text data. This has been extended to support searching of XML documents, including section and subsection searches.

*inter*Media Text Example: Searching Text and XML Data Using CONTAINS

This *inter*Media Text example presume you have already created the appropriate index.

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

> **See Also:** Chapter 5, "Using interMedia Text to Search and Retrieve Data from XML Documents" for more information on *inter*Media Text.

XML Document Unstructured Storage ("Authored XML"): Advantages

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

XML Document Unstructured Storage ("Authored XML"): Disadvantages

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

Use a Hybrid Storage Approach for Better Mapping Granularity

In the previous section we discussed the following:

- How structured XML documents can be mapped to object-relational instances
- How unstructured XML documents can be stored in LOBs

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

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

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

A Hybrid Approach Allows for User-Defined Granularity for Storage

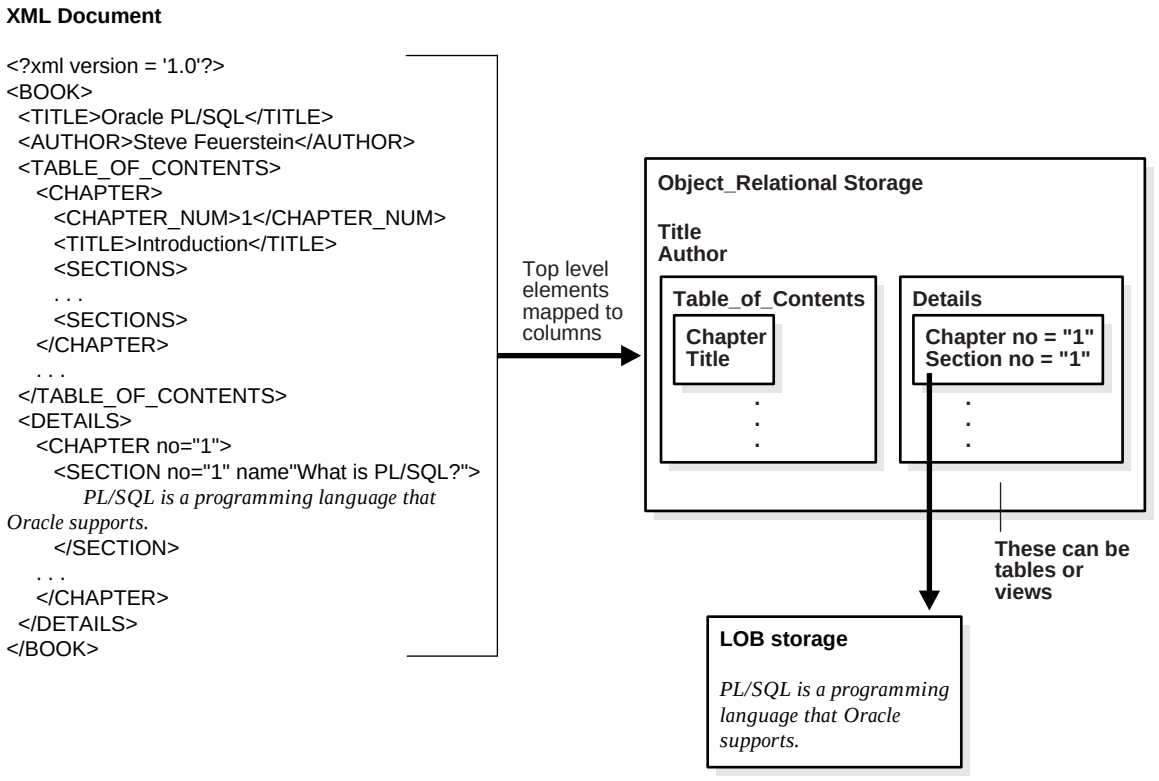
The alternative is to have user-defined granularity for such storage. In the book example, you may want the following:

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

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

Figure 1–3 illustrates the Hybrid approach to storage.

Figure 1–3 Hybrid Approach: Querying Top Level Elements in Tables While Contents are in a CLOB



Hybrid Storage Advantages

The advantage of the hybrid storage approach for storing XML documents, is the following:

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

Generated XML: XML Transformations

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

Transforming Query Results Using XSLT

This involves querying on the original document and transforming the result into a form required by the end-user or application. For instance, if an application is talking to a cellular phone using WML, it might need to transform the XML generated into WML or other similar standard suitable for communicating with the cellular phone.

This can be accomplished by applying XSLT transformations on the result ing XML document. The XSLT transformations can be pushed into the generation phase itself as an optimization. A scalable, high performance XSLT transformation engine within the database server would be able to handle large amounts of data.

XML Schemas and Mapping of Documents

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

- XML Schema1: Constraining document structure (elements, attributes, namespaces)
- XML Schema2: Constraining content (datatypes, entities, notations)

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

> **Note:** The XML Schema API is not provided with the software of this release. However, it is available for download from OTN.

XML Schema provides inheritance for element, attribute, and datatype definitions. Mechanisms are provided for URI references to facilitate a standard, unambiguous

semantic understanding of constructs. The schema language also provides for embedded documentation or comments.

For example, you can define a simple data type as shown in "XML Schema Example 1: Defining a Simple Data Type".

XML Schema Example 1: Defining a Simple Data Type

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

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

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

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

XML Schema Example 2: Using XML Schema to Map Generated XML Documents to Underlying Schema

Consider the simple Purchase Order type expressed in XML Schema:

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

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

```
   <element name="custAddr" type="Address" />
</type>

<type name="Items">
   <element name="lineItem" minOccurs="0" maxOccurs="*">
    <type>
      <element name="lineItemNo" type="positiveInteger" />
      <element name="lineItemName" type="string" />
      <element name="lineItemPrice" type="number" />
      <element name="LineItemQuan">
        <datatype basetype="integer">
          <minExclusive>0</minExclusive>
        </datatype>
      </element>
    </type>
   </element>
</type>

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

These XML Schemas have been deliberately constructed to match closely the Object-Relational purchase order example described above in"Running the above code results in the following XML document:" on page 1-24.

The point is to underscore the close match between the proposed constructs of XML Schema with SQL:1999-based type systems. With such a close match, it is easy for us to map an XML Schema to a database Object-Relational schema, and map documents that are schema-valid according to the above schema to row objects in the database schema. In fact, the greater expressiveness of XML Schema over DTDs greatly facilitates the mapping.

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

A simple example might be a memo, which is routed differently based on markup tags. A more sophisticated example is a technical service manual for an intercontinental aircraft. Based on complex constraints provided by XML Schema, you can ensure that the author of such a manual always enters a valid part-number,

and you might even ensure that part-number validity depends on dynamic considerations such as inventory levels, fluctuating demand and supply metrics, or changing regulatory mandates.

"Authored XML": Storing XML Documents as Documents

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

Authored XML Examples

Examples include written text such as articles, advertisements, books, legal contracts, and so on.

Documents of this nature are known as document-centric and are delivered from the database as a whole. Storing this kind of document intact within Oracle8*i* gives you the advantages of an industry-proven database and its reliability over file system storage.

Storage Outside the Database

If you choose to store an XML document outside the database, you can still use Oracle8*i* features to index, query, and efficiently retrieve the document through the use of BFILES, URLs, and text-based indexing.

Generating and Storing XML Documents

If the XML document has a well-defined structure and contains data that is updateable or used in other ways, the document is data-centric. Typically, the XML document contains elements or attributes that have complex structures.

Generated XML Examples

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

Oracle8*i*, with its object-relational extensions has the ability to capture the structure of the data in the database using object types, object references, and collections. There are two options for storing and preserving the structure of the XML data in an object-relational form:

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

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

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

When the XML Document Structure Needs Transforming

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

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

Combining XML Documents and Data Using Views

Finally, if you have a combination of structured and unstructured XML data, but still want to view and operate on it as a whole, you can use Oracle*8i* views.

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

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

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

How to Generate a Web Form's Infrastructure

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

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

XML-SQL Utility Storage

If you use XML-SQL Utility to store XML data, it can map XML into any table or view. It canonically maps elements to and from columns.

- Advantages
  - Can handle objects, LOBs and all Oracle types
  - Object views can map any structured documents into multiple tables

- Disadvantages
  - Provides only canonical mapping back, but does allow registering of XSLTs; thus transforming the XML document on the fly as it generates it.

General XML Design Issues for Data Exchange Applications

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

- Use a Hybrid Storage Approach for Better Mapping Granularity
- Sending XML Data from a Web Form to a Database
- Communicating XML Documents Among Applications

Use a Hybrid Storage Approach for Better Mapping Granularity

This was already discussed in "Use a Hybrid Storage Approach for Better Mapping Granularity" on page 1-30.

Figure 1–3 illustrates the Hybrid approach to storage.

Sending XML Data from a Web Form to a Database

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

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

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

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

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

Communicating XML Documents Among Applications

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

In this discussion:

- The sending application transmits the XML document
- The receiving application receives the XML document
- *File Transfer*. The receiving application requests the XML document from the sending application via FTP, NFS, SMB or other file transfer protocol. The document is copied to the receiving application's file system. The application reads the file and processes it.
- *HTTP*. The receiving application makes an HTTP request to a servlet. The servlet returns the XML document to the receiving application, which reads and processes it.
- *Web Form*. The sending application renders a web form. A user fills out the form and submits the information via a Java applet or Javascript running in the browser. The applet or Javascript transmits the user's form in XML format to the receiving application, which reads and processes it. If the receiving application will ultimately write data to the database, the sending application should create the XML in a database compatible format. One way to do this using Oracle XML products is described in the section Sending XML Data from a Web Form to a Database.
- *Advanced Queuing*. An Oracle8i database sends an XML document via Net8 and JDBC to the one or more receiving applications as a message through Oracle Advanced Queueing (AQ). The receiving applications dequeue the XML message and process it.

  See Also:

  - Chapter 9, "Using Oracle Advanced Queuing (AQ) in XML Data Exchange"
  - Chapter 13, "B2B XML Application: Step by Step"
  - Oracle Integration Server Overview

Oracle XML Samples and Demos

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

What's Needed to Run Oracle XML Components

Oracle8*i* includes native support for Internet standards, including Java and XML. You can run Oracle XML components and applications built with them *inside* the database itself using Oracle JServer, Oracle8*i*'s built-in Java Virtual Machine.

Use Oracle8*i* Lite to store and retrieve XML data, for devices and applications that require a smaller database footprint.

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

Requirements

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

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

Requirements are also discussed in the XDK chapters and Appendixes.

XML Technical Support is Free on OTN

Technical support for Oracle XML platform and utilities is available for free via the XML Discussion Forum on Oracle Technical Network (OTN):

http://technet.oracle.com/tech/xml

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

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

Download the Latest Software From OTN

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

http://technet.oracle.com/software/tech/xml

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

What's New with Oracle and XML Schema, XML Query, XLink, and Xpointer?

For the latest information about XML Schema, XML Query, XLink, and XPointer go to the OTN XML site:

http://technet.us.oracle.com/tech/xml

2

Business Solutions Using Oracle XML

This chapter contains the following sections:

Applications that Use Oracle XML

There are many potential uses for XML in Internet applications.

Two database-centric application areas where Oracle XML components are well suited are:

- "Content and Document Management", including customizing data presentation
- "Business-to-Business / Consumer (B2B / B2C) Messaging" for data exchange among inter or intra system applications

and any combinations of these. This manual focuses on these two application areas. Some typical scenarios in each of these two areas have been identified and are described here.

Content and Document Management

Customizing Presentation of Data

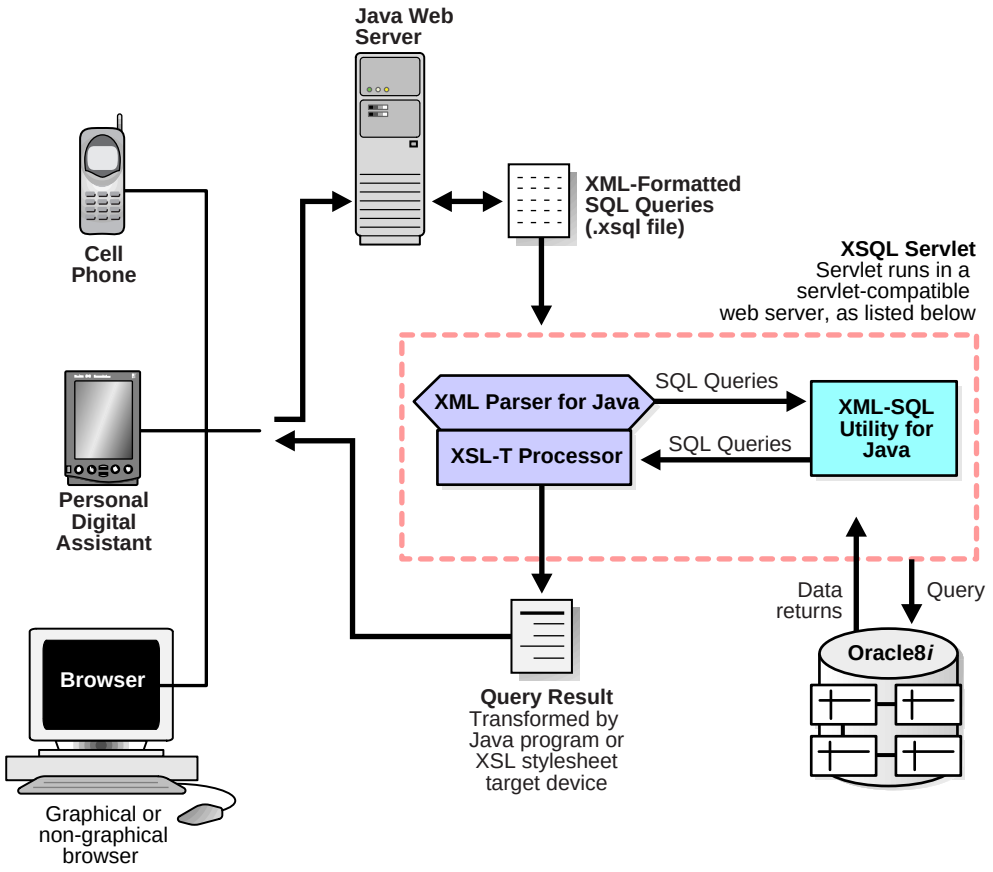
XML is increasingly used to enable customized presentation of data for different browsers, devices, and users. By using XML documents along with XSL stylesheets on either the client, middle-tier, or server, you can transform, organize, and present XML data tailored to individual users for a variety of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

- Graphical and non-graphical web browsers
- Personal digital assistants (PDAs) like the Palm Pilot
- Digital cell phones and pagers
- TBD... and others.

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

Using XML and XSL also makes it easier to create and manage dynamic web sites. You can change the look and feel simply by changing the XSL stylesheet, without having to modify the underlying business logic or database code. As you target new users and devices, you can simply design new XSL stylesheets as needed. This is illustrated in Figure 2–1.

Figure 2–1 Content Management: Customizing Your Presentation



See Also:

- Chapter 7, "Personalizing Data Display With XML: Portal-to-Go"
- Chapter 17, "Using XML Parser for Java"

Consider the following content management scenarios:

- Scenario 1. Content and Document Management: Creating and Publishing Composite Documents Using Oracle XML
- Scenario 2. Content and Document Management: Delivering Personalized Information Using Oracle XML
- Scenario 3. Content Management: Using XML to Customize Data Driven Applications

These scenarios use Oracle XML components. The scenarios include a description of each business problem, solution, main tasks, and Oracle XML components used.

These scenarios are further illustrated with several case studies in Part 3.

Scenario 1. Content and Document Management: Creating and Publishing Composite Documents Using Oracle XML

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

Solution. Company X can use XSL stylesheets to assemble the document sections or fragments and deliver the composite documents electronically to users. See Figure 2–2.

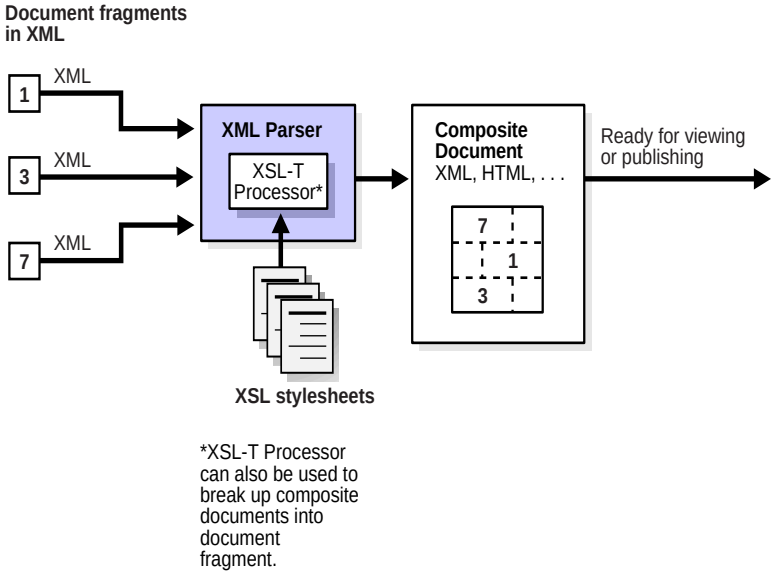
Main Tasks Involved. These are the main tasks involved in Scenario 1:

1. Break up the documents into desired small usable sections or fragments.
2. Store these sections or fragments in CLOBs and BLOBs in the database.
3. Create XSL Stylesheets to render the sections or fragments into complete documents

Oracle XML Used:

- XML Parser with XSL-T
- XSU to move sections or fragments into and out of the database

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



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

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

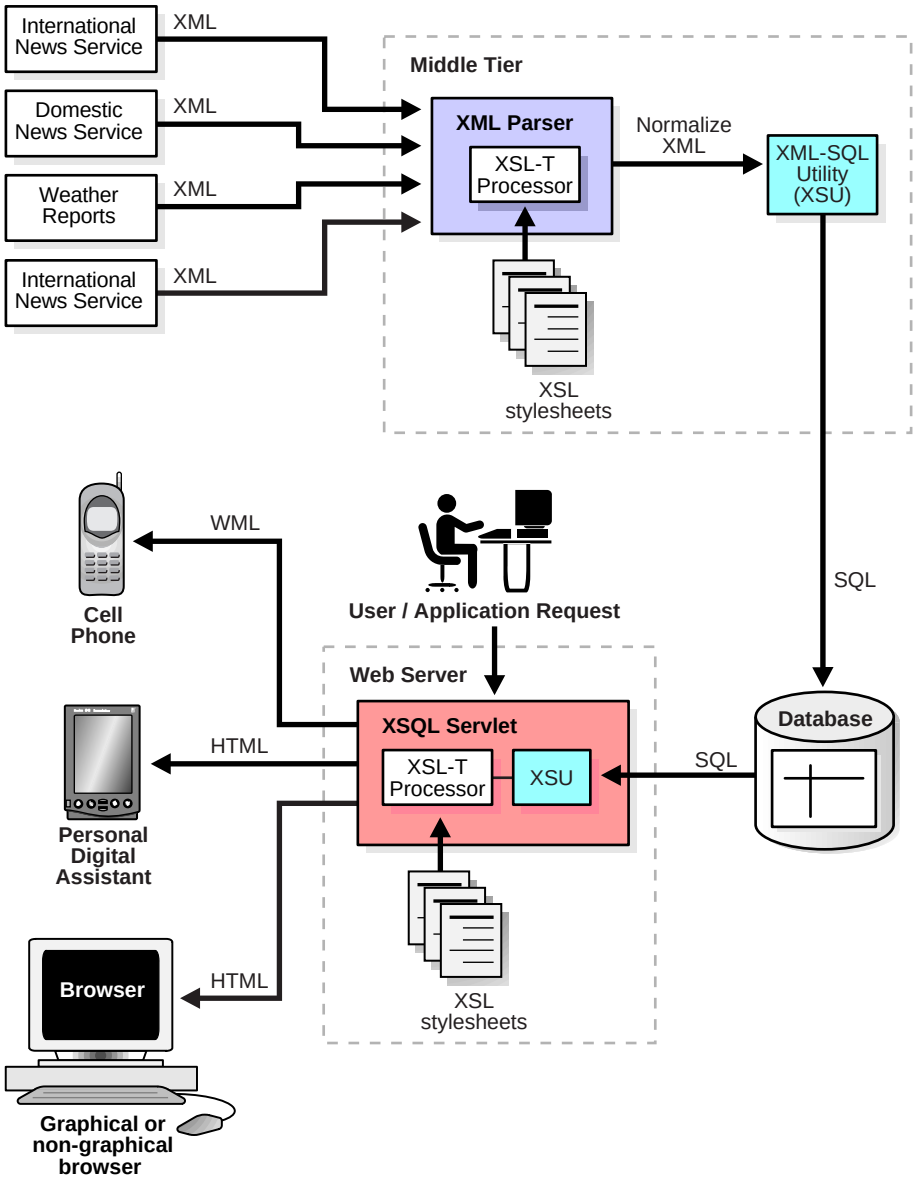
- *Solution*: Use XSL stylesheets in conjunction with the XSQL Servlet to dynamically deliver appropriate rendering to the requesting service. See Figure 2–3.
- *Main Tasks Involved*. These are the main tasks involved in Scenario 2:

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

  See also Chapter 6, "Customizing Content with XML: Dynamic News Application".

- *Oracle XML Used:*
  - XML Parser for Java v2
  - XML-SQL Utility (XSU)
  - XSQL Servlet

Figure 2–3 Scenario 2. Using XML to Deliver Customized News Information



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

Problem. Company X needs data and interactivity delivered to a thin client.

- *Solution*. Data is queried from a database and rendered dynamically through one or more XSL stylesheets for sending to the client application. The data is stored in a relational database in LOBs and materialized in XML.
- *Main Tasks Involved:* See Chapter 11, "Customizing Discoverer 3i Viewer with XSL".
- *Oracle XML Used:*
  - XML Parser for Java

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

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

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

Business-to-Business/Consumer (B2B/B2C) Messaging With XML: Scenarios

Consider the following business-to-business / consumer (B2B / B2C) messaging scenarios:

- Scenario 4. B2B Messaging: Online Multivendor Shopping Cart Design Using XML
- Scenario 5. B2B Messaging: Using XML and Oracle Advanced Queueing for an Internet Application
- Scenario 6. B2B Messaging: Using XML and Oracle Advanced Queueing Messaging for Multi-Application Integration

These scenarios use Oracle XML platform components. Each of these is described briefly including the problem at hand, the solution, and main tasks used to resolve the problem.

These scenarios are further illustrated with detailed case studies in Parts 4, and 5.

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

Problem. Company X needs to build an online shopping cart the products on which come from various vendors. Company X want to receive orders online and then according to what product is ordered funnel the order to the correct vendor accordingly.

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

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

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

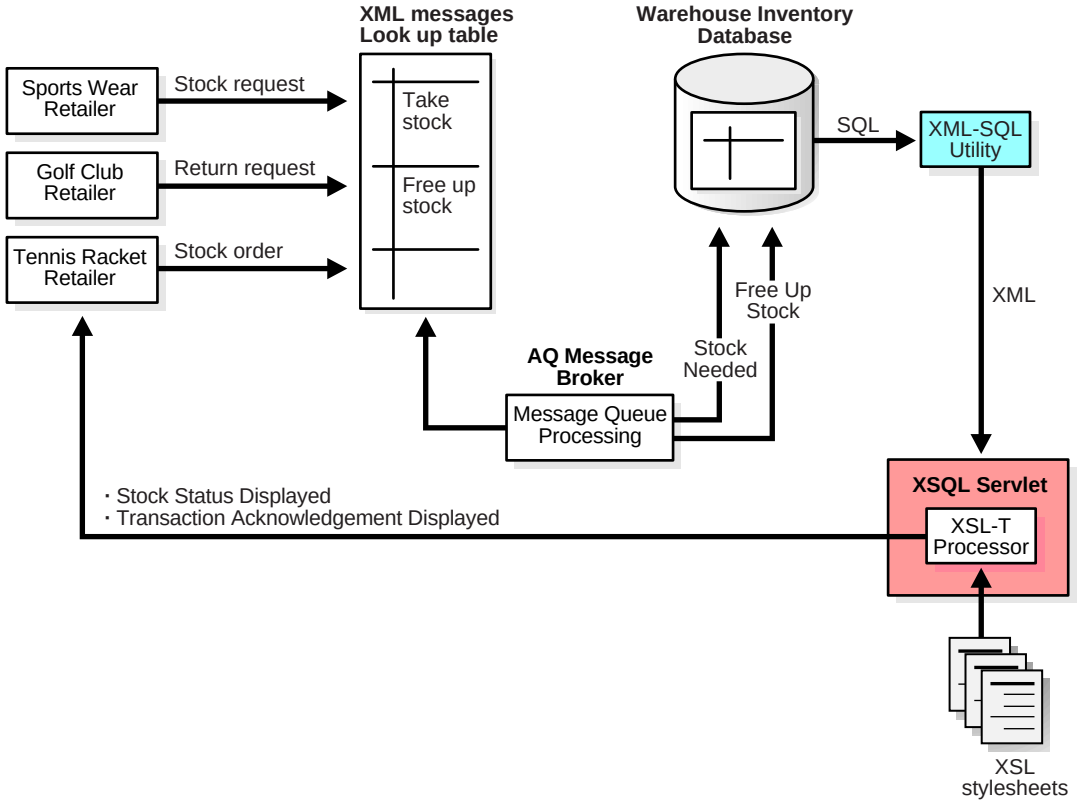
  Customer orders are compiled into XML and delivered to the appropriate vendor databases for processing. XSL is used to transform and divide the cart for compliant transfers. Data is stored in a relational database and materialized using XML. See Figure 2–4.

- *Main Tasks Involved*. For examples of similar implementations see:

  - Chapter 10, "B2B: How iProcurement Uses XML to Offer Multiple Catalog Products to Users"
  - Chapter 13, "B2B XML Application: Step by Step"

- *Oracle XML Components Used*: Oracle XML Parser, *XML-SQL Utility*, XSQL Servlets, XML-SQL Utility, and custom Java code for authenticating the "Shopping Cart" it receives from the vendor web sites.

Figure 2–4 Scenario 4. Online Multivendor Shopping Cart Design Using XML

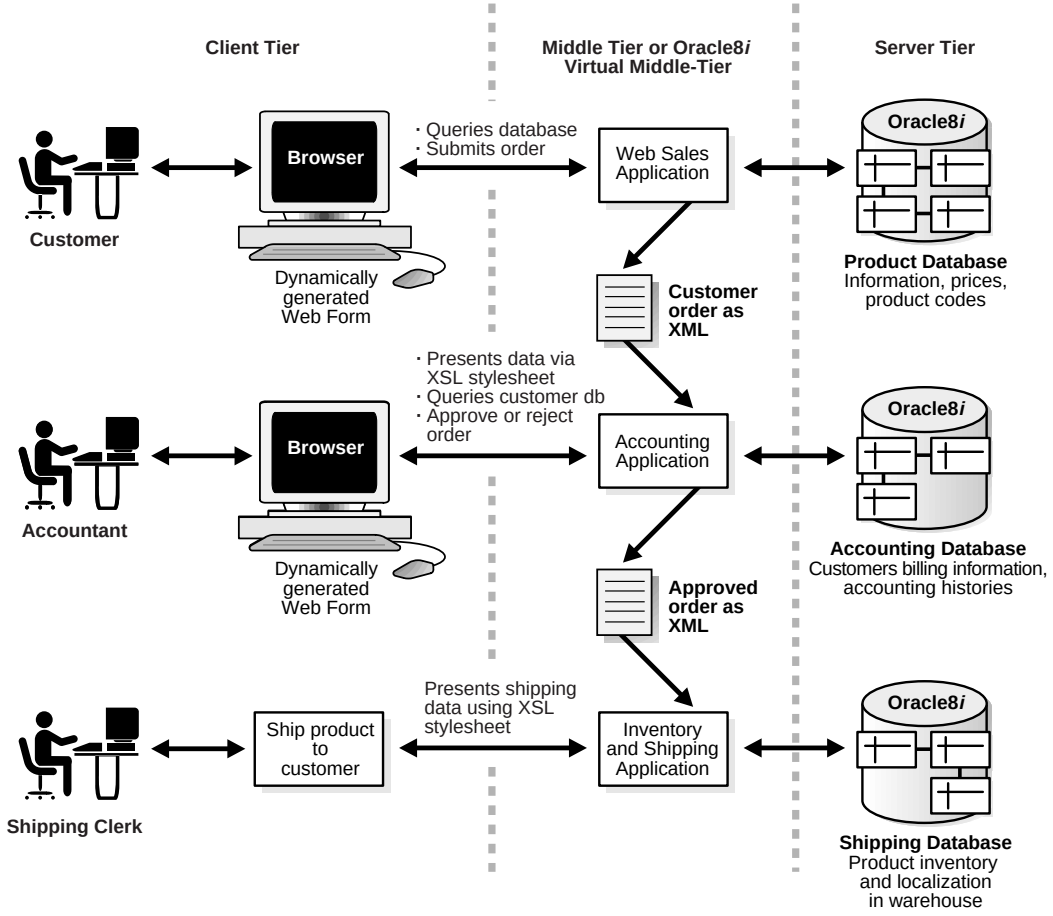


Scenario 5. B2B Messaging: Using XML and Oracle Advanced Queueing for an Internet Application

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

- *Solution*. When a resource is accessed or released this triggers an 'availability XML message'. This in turn transforms the resource, using XSL, into multiple client formats according to need. Conversely, a resource acquisition by one client sends an XML message to other clients, hence signalling removal. Messages are stored in LOBs. Data is stored in a relational database and materialized in XML. See Figure 2–5.
- *Main Tasks Involved*. See also Chapter 12, "Phone Number Portability Using XML Messaging".
- *Oracle XML Used:*
  - XML Parser
  - XSL-T Processor

Figure 2–5 Scenario 5. Using XML and Oracle Advanced Queueing Messaging for an Internet B2B Application

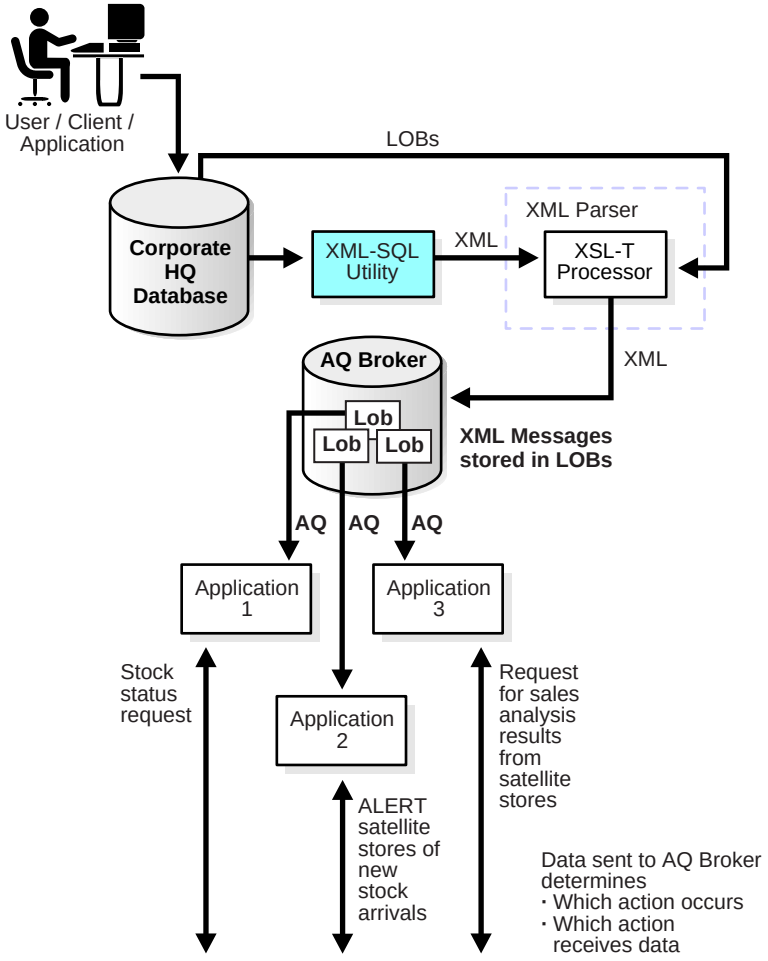


Scenario 6. B2B Messaging: Using XML and Oracle Advanced Queueing Messaging for Multi-Application Integration

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

- *Solution*. XML is used as the message payload. It is transformed via the XSL-T Processor, enveloped and routed accordingly. The XML messages are stored in an 'AQ Broker' Database in LOBs. This solution also utilizes content management i.e., presentation customization using XSL stylesheets. See Figure 2–6.
- *Main Tasks Involved.*
  1. The user or application places a request. The resulting data is pulled from the corporate database using XSU.
  2. Data is transformed via XSL-T Processor and sent to the AQ Broker
  3. AQ Broker reads this message and determines accordingly, what action is needed. It issues the appropriate response to (or from) from Application 1, 2, and 3, for further processing.
- *Oracle XML Used:*
  - XML Parser
  - XSL-T Processor
  - XML-SQL Utility (XSU)

Figure 2–6 Scenario 6. Using XML and Oracle Advanced Queueing Messaging for Multi-Application Integration



3

Oracle XML Components and General FAQs

This chapter contains the following sections:

Oracle XML Components: Overview

Oracle*8i* provides several components, utilities, and interfaces you can use to take advantage of XML technology in building your web-based database applications. Which components you use depends on your application requirements, programming preferences, development, and deployment environments.

The following XML components are provided with Oracle8i:

- **XML Developer's Kits (XDKs)**[1]. There are Oracle XDKs for Java, JavaBeans, C, C++, and PL/SQL. These development kits contain building blocks for reading, manipulating, transforming, and viewing XML documents.
- **XML-SQL Utility (XSU)**. XML-SQL Utility for Java and PL/SQL: Generates (gets) and stores (puts) 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.

Table 3–1 lists the Oracle XML components and associated languages.

Table 3–1 Languages Available for XDK and XSU

Language	Java	C	C++	PL/SQL
Parser	Yes	Yes	Yes	Yes
XSLT Processor	Yes	Yes	Yes	Yes
Class Generator	Yes	---	Yes	---
XSQL	Yes	N/A	N/A	N/A
Transviewer Beans	Yes	N/A	N/A	N/A
XML-SQL Utility	Yes			Yes

This chapter provides an overview of the XML components.

Development Tools and Other XML-Enabled Oracle8*i* Features

- Oracle*8i inter*Media Text: An application interface in Java where data can be viewed as documents and documents can be treated as data.

[1] Oracle XDK is fully supported and comes with a commercial redistribution license.

- Oracle JDeveloper: An integrated development tool for building Java web-based applications
- Oracle*8i* Internet File System (*i*FS): An application interface in Java where data can be viewed as documents and the documents can be treated as data.

XDK for Java

XDK for Java is comprised of the following:

- **XML Parser for Java:** Creates and parses XML using industry standard DOM and SAX interfaces. Includes an **XSL Transformation (XSLT) Processor** that transforms XML to XML or other text-based formats, such as, HTML.
- **XML Java Class Generator:** Generates Java classes.
- **XML Transviewer Java Beans:** View and transform XML documents and data via Java
- **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.

XDK for C

XDK for C is comprised of the following:

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

XDK for C++

XDK for C++ is comprised of the following:

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

XDK for PL/SQL

XDK for PL/SQL is comprised of the following:

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

XML Parsers

Oracle's XML parser includes implementations in C, C++, PL/SQL, and Java for the full range of platforms on which Oracle*8i* runs.

Based on conformance tests, xml.com published Oracle's parser ranked 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 1.0.

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

- XPath[1]
- Incremental XSL transformation of document nodes. XSL transformations are compliant with version 1.0 of the W3C recommendations. This support enables the following:
  - Transformations of XML documents to another XML structure
  - Transformations of XML documents to other text-based formats

The parsers are available on all Oracle platforms.

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

[1] XPath is the W3C recommendation that specifies the data model and grammar for navigating an XML document utilized by XSLT, XLink and XML Query.

See Also: Chapter 17, "Using XML Parser for Java" and Appendix C, "XDK for Java: Specifications and Cheat Sheets".

Figure 3–1 The Oracle XML Parser for Java

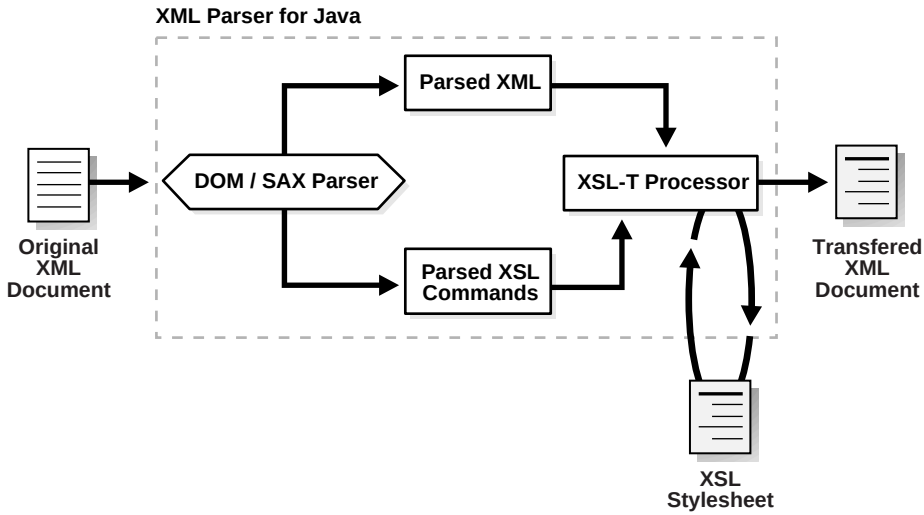
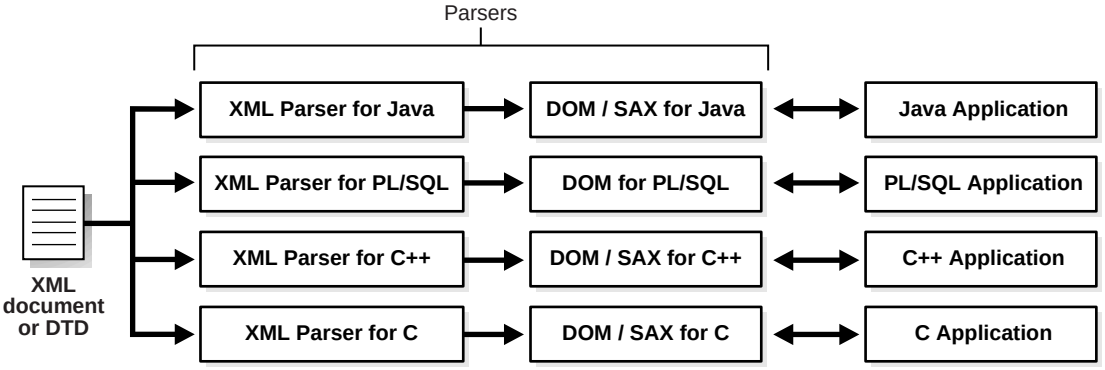


Figure 3–2 The XML Parsers: Java, C, C++, PL/SQL



XSL Transformation (XSLT) Processor

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

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

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 17, "Using XML Parser for Java".

> **See Also:** Appendix C, "XDK for Java: Specifications and Cheat Sheets".

XML Class Generator

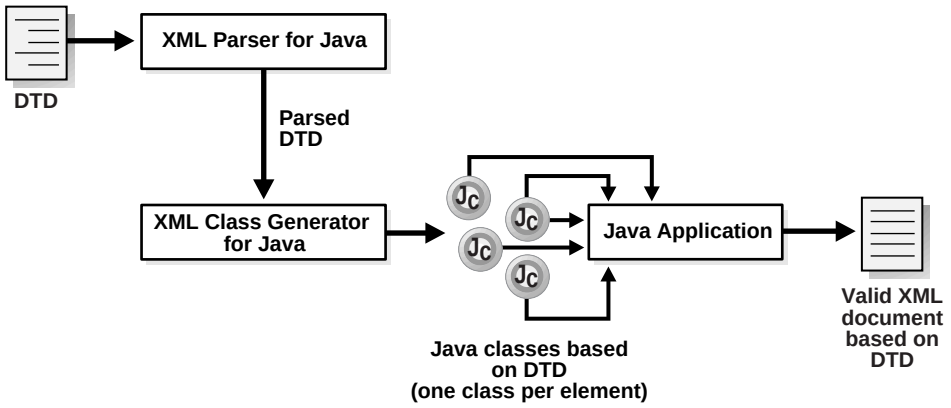
The XML Class Generator creates a set of Java or C++ classes for creation of XML documents conformant with the input DTD.

Figure 3–3 shows the Oracle XML Class Generator overall functionality.

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

- Chapter 18, "Using XML Java Class Generator"
- Chapter 23, "Using XML C++ Class Generator"

Figure 3–3 Oracle XML Java Class Generator



XML Transviewer Java Beans

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

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

- **DOM Builder Bean**. This wraps the Java XML (DOM) parser with a bean interface, allowing multiple files to be parsed at once (asynchronous parsing). By registering a listener, Java applications can parse large or successive documents having control return immediately to the caller.
- **XML Source Viewer Bean**. This bean extends JPanel by enabling the viewing of XML documents. It improves the viewing of XML and XSL files by color-highlighting XML and XSL syntax. This is useful when modifying an XML document with an editing application. Easily integrated with the DOM Builder Bean, it allows for pre 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 optionally edit them
  - Transformer bean to apply the stylesheet to the XML document and view the output

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 3, "Oracle XML Components and General FAQs".

Oracle XSQL Page Processor and Servlet

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

You can use XSQL Servlet to perform the following tasks:

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

Servlet Engines that Support XSQL Servlet

XSQL Servlet has been tested with the following servlet engines:

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

JavaServer Pages (JSP) Platforms that Support XSQL Servlet

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 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)
- Oracle8i Lite Web-to-Go Server with Oracle JSP 1.0
- Oracle8i 8.1.7 Beta "Aurora" Servlet Engine with Oracle JSP 1.0
- Any Servlet Engine with Servlet API 2.1+ and Oracle JSP 1.0

In general, it should work with the following:

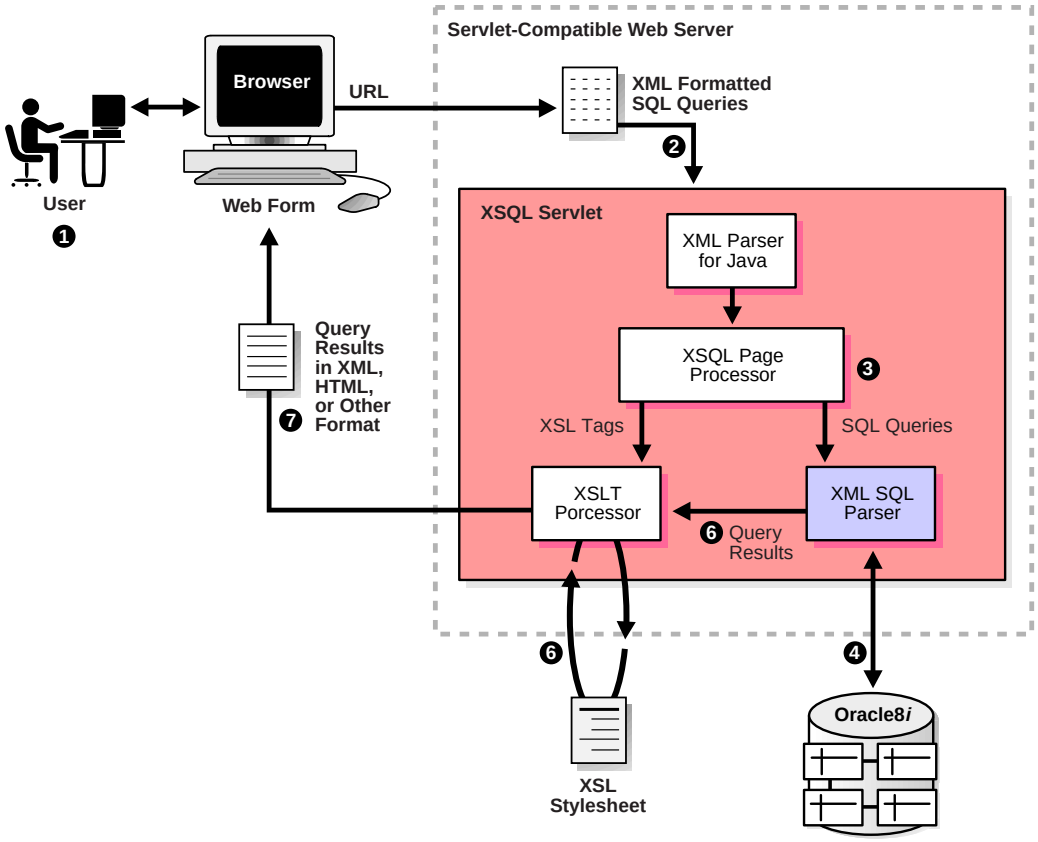
- Any servlet engine supporting the Servlet 2.1 Specification or higher
- Oracle JSP 1.0 reference implementation or functional equivalent from another vendor

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

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

1. The user enters a URL through a browser, which is interpreted and passed to the XSQL Servlet through a Java Web Server. The URL contains the name of the target XSQL file (.xsql) and optionally, parameters, such as values and an XSL stylesheet name. Alternatively, the user can invoke the XSQL Servlet from the command line, bypassing the browser and Java web server.
2. The servlet passes the XSQL file to the XML Parser for Java, which parses the XML and creates an API for accessing the XML contents.
3. The page processor component of the servlet uses the API to pass XML parameters and SQL statements (found between <query></query> tags) to XML-SQL Utility. The page processor also passes any XSL processing statements to the XSLT Processor.
4. XML-SQL Utility sends the SQL queries to the underlying Oracle*8i* 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.

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



Oracle XML-SQL Utility (XSU)

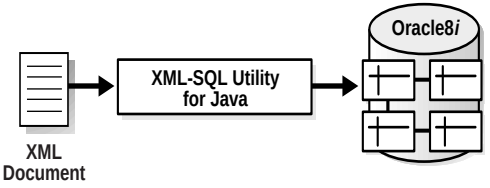
Oracle XML-SQL Utility (XSU) supports Java and PL/SQL.

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

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

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

Figure 3–5 Oracle XML-SQL Utility Functional Diagram



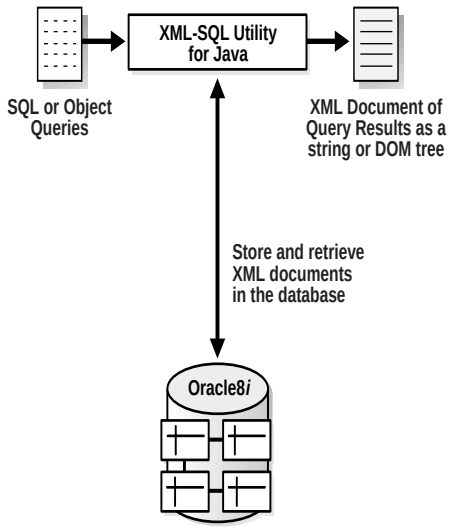
XML SQL Utility for Java consists of a set of Java classes that perform the following tasks:

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

Generating XML from Query Results

Figure 3–6 shows how XML SQL Utility processes SQL queries and returns the results as an XML document.

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



XML Document Structure: Columns Are Mapped to Elements

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

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

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

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

- A string representation of the XML document. Use this respresentation 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 4, "Using XML-SQL Utility (XSU)" and Chapter 13, "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 4, "Using XML-SQL Utility (XSU)".

Oracle *inter*Media Text

*inter*Media Text extends Oracle8*i* by indexing any text or documents stored in Oracle8*i*.

You can use *inter*Media Text to perform searches on XML documents stored in Oracle8*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:** Chapter 5, "Using interMedia Text to Search and Retrieve Data from XML Documents", for more information on using *inter*Media Text and XML.

Tools for Building Oracle XML Applications

Tools you can use to develop Oracle XML applications are:

- Jdeveloper. See Chapter 14, "Using JDeveloper to Build Oracle XML Applications"
- Internet file System *(i*FS). See Chapter 15, "Using Internet File System (iFS) to Build XML Applications"

Oracle XML Components: Generating XML Documents

Figure 3–7 through Figure 3–10 illustrate the relationship of the Oracle XML components and how they work together to generate XML documents from Oracle8*i* via an SQL query. The options are depicted according to language used:

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

Using Oracle XML Components to Generate XML Documents: Java

Figure 3–7 shows the Oracle XML Java components and how they can be used to generate an XML document. Available XML Java components are:

- XDK for Java:
  - XML Parser for Java, Version 2 including the XSLT
  - XML Class Generator
  - 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:

A. By the XSL Servlet (this includes using XSU and XML Parser)

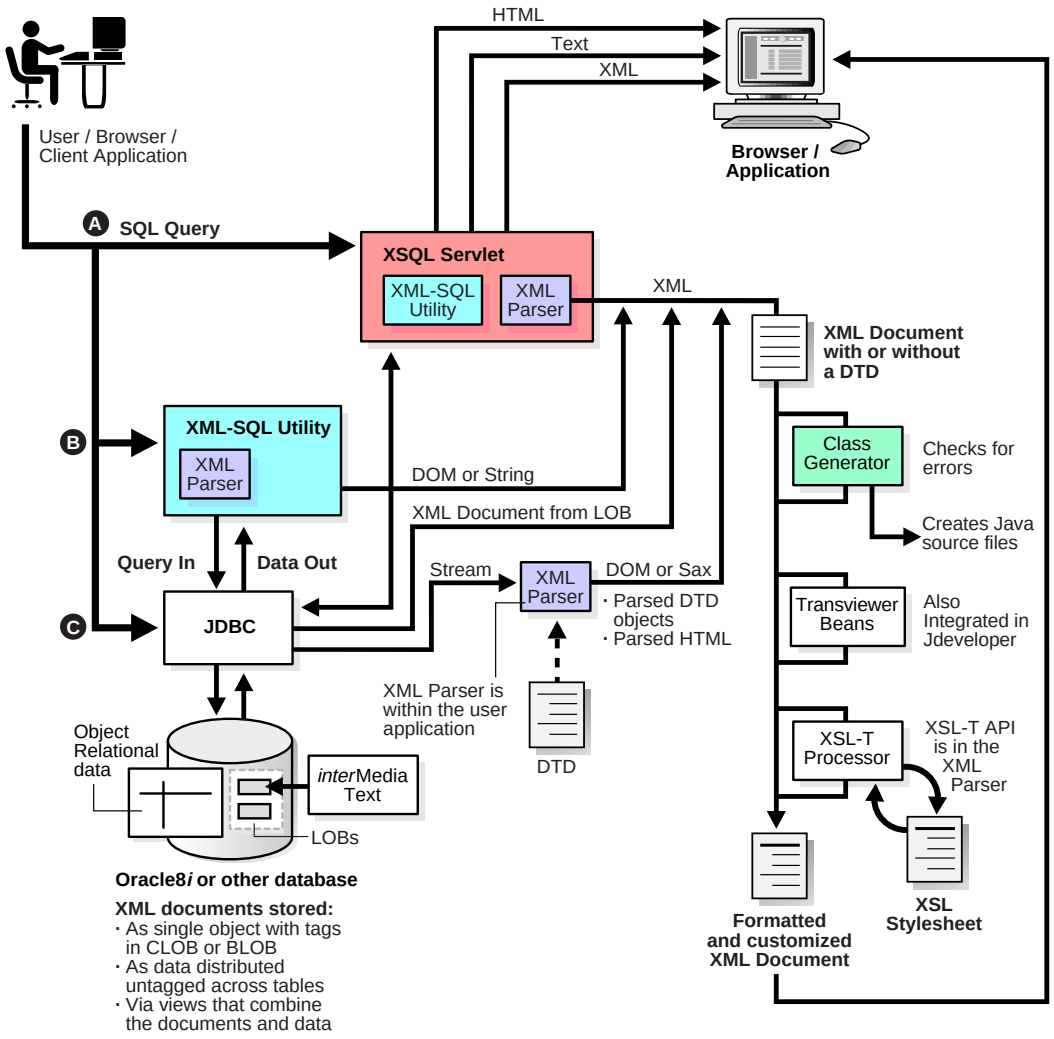
B. Directly by the XSU (this includes XML Parser)

C. Directly by JDBC which then accesses XML Parser

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

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

Figure 3–7 Using Oracle XML Components to Generate an XML Document - Java Options



Using Oracle XML Components to Generate XML Documents: C

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

- XML Parser / XSLT Processor

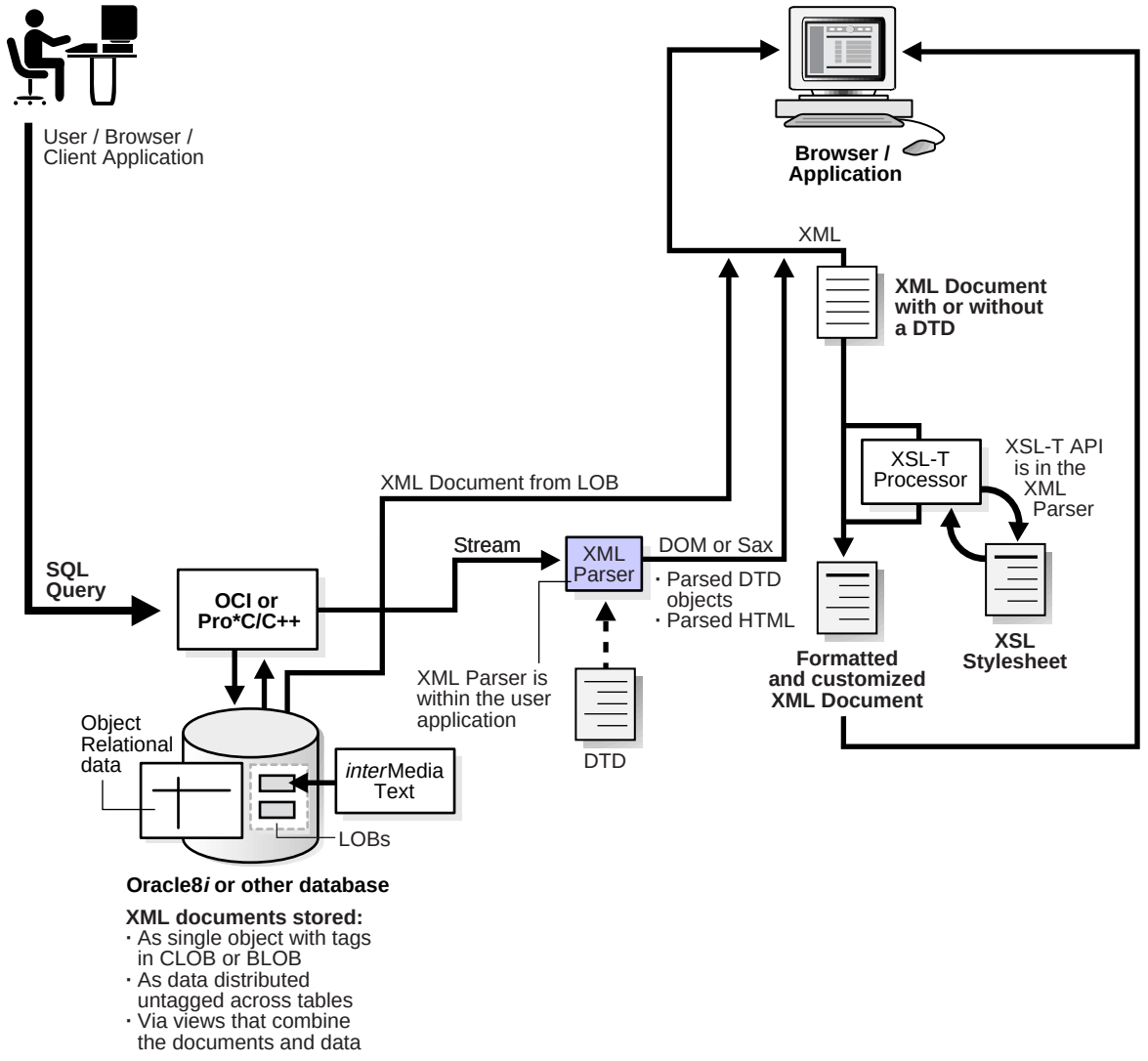
Your SQL queries are sent to the database via OCI or as embedded statements in the Pro*C precompiler.

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

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

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

Figure 3–8 Using Oracle XML Components to G enerate an XML Document - C Options



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

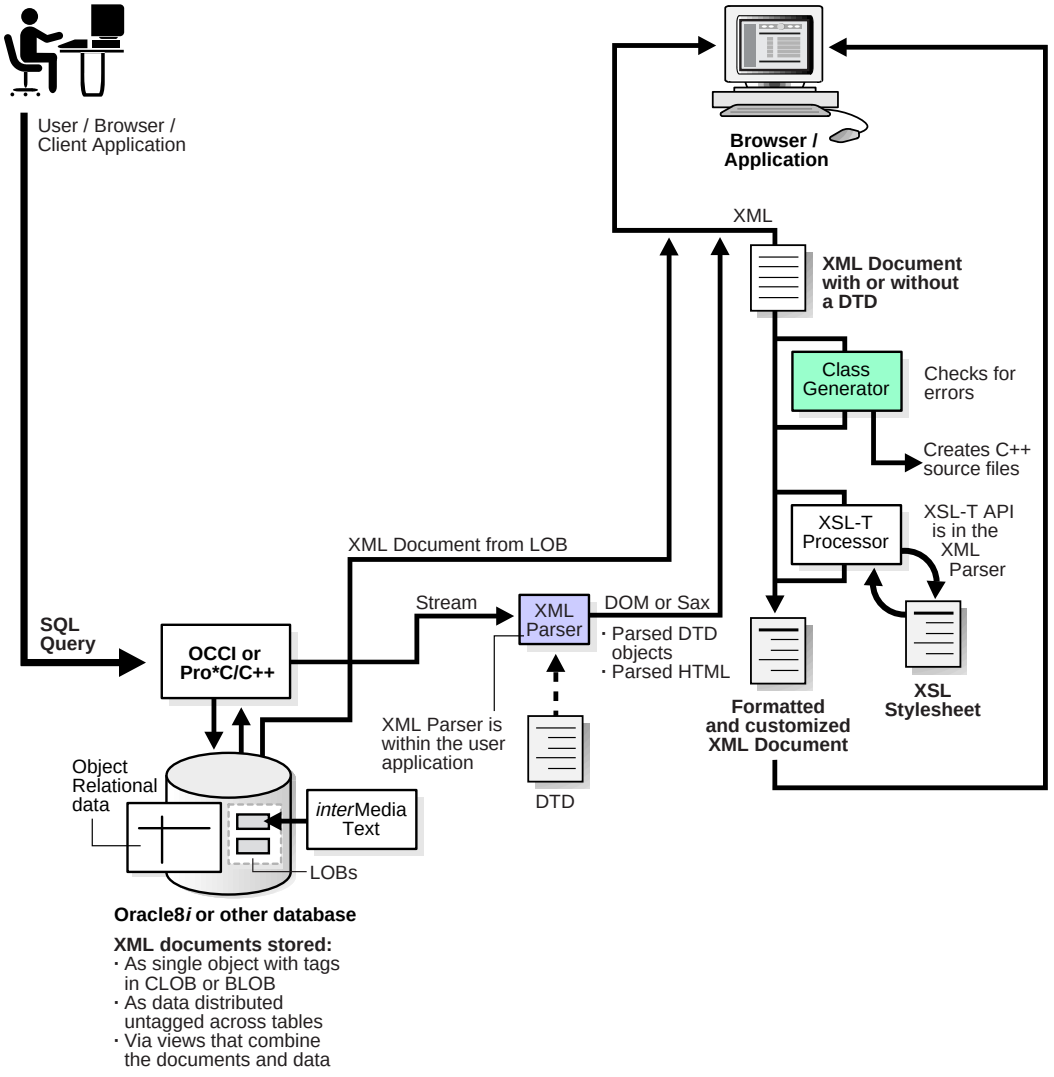
Figure 3–9 shows the Oracle XML C++ components used to generate an XML document. Available XML C++ components are:

- XDK for C++:
  - XML Parser for C++, Version 2 including the XSLT
  - XML C++ Class Generator

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

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

Figure 3–9 Using Oracle XML Components to G enerate an XML Document - C++ Options



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

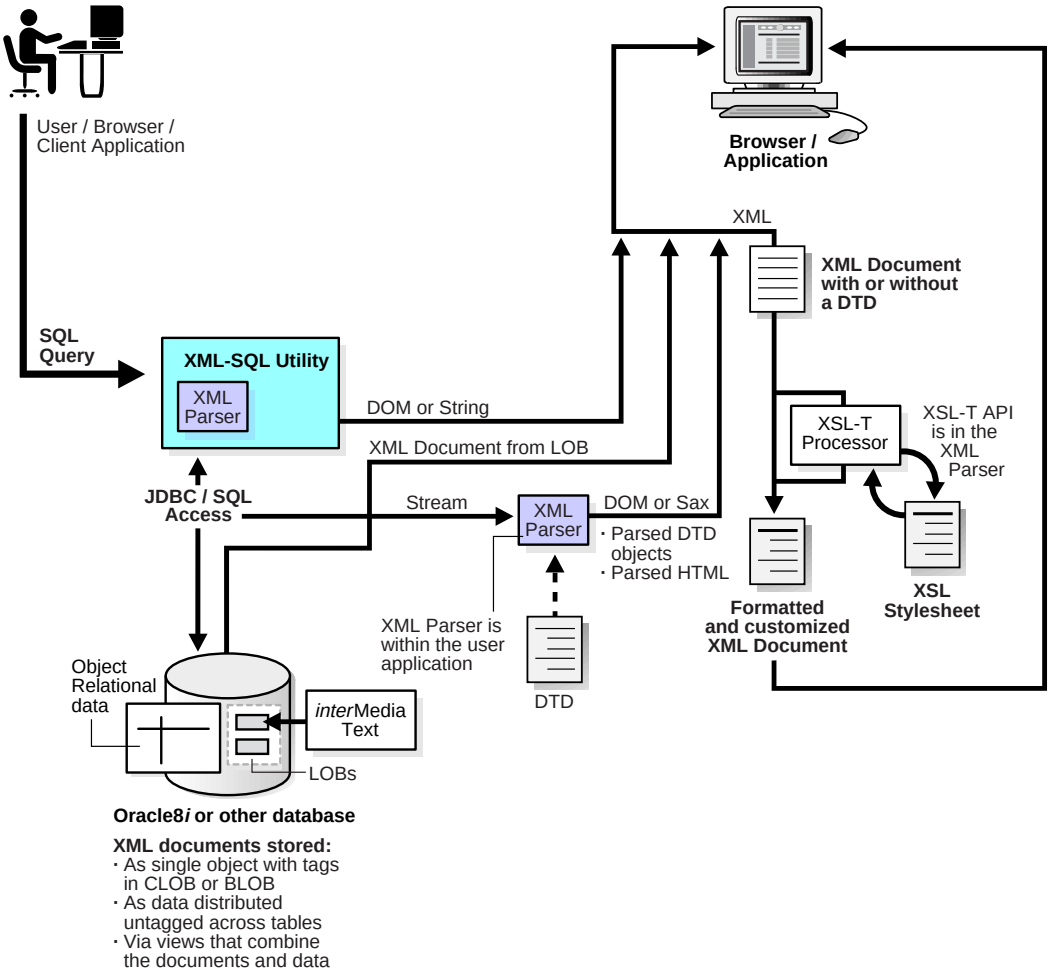
Figure 3–10 shows the Oracle XML PL/SQL components used to generate an XML document. Available XML PL/SQL components are:

- XDK for PL/SQL:
  - 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 query (SQL), there are two possible ways of processing the query using the Oracle XML components:

- Directly via JDBC which then accesses the XML Parser
- Via XML-SQL Utility

Figure 3–10 Using Oracle XML Components to Generate an XML Document - PL/SQL Options



Frequently Asked Questions (FAQs) - General XML

This section includes general FAQs about Oracle XML technology.

There are also FAQs at the end of many chapters in the manual.

How do I Start Writing XML?

Question

I read some articles about XML, but how do I start writing it. If it is C,

1. in unix I open vi

2. include header files

3. open main

4. write some code

5. close main

6. compile (cc -o example example.c)

How do we do this in XML? XML is discussed in many journals, but there is no mention of how to start.

Answer

The easiest way to get started is to use the Oracle XSQL Servlet to leverage your understanding of SQL to begin experimenting with XML and XSLT Transformations. If you wait a day or so, a new release (0.9.8.6) with lots more tutorial info will be available, but the current release (0.9.6.2) is good to get started with and has extensive release notes to help you get started.

See our http://technet.oracle.com/tech/xml page and click on "Oracle XSQL Servlet".

XML and Required Oracle Tools

Question

I am exploring options to use Oracle's XML technology for generating data files. The generated data (XML) files would then be converted to "client specific" EDI text files using a "translator".

The translator would be used until such time that the client is ready to accept XML documents.

1. What is the latest version of database II need to use? We have 8.0.6. Do we need Oracle8i?
2. Is there a translator already available to convert XML to text files? Is XSL such a tool?
3. Where and how would the translation mapping be stored?

Answer

You only need Oracle8i if you would like to generate the XML directly from the database instead of using a middle tier.

You can make Java calls to XML- SQL Utility from PL/SQL.

XML files are text files, but you can use XSL to transform the XML to almost any text-based format by creating the appropriate stylesheet.

Collecting Purchase Orders in XML: Creating an RFP in XML?

Question

I am going to develop a small application using XML and Oracle8i. Here is the scenario...Company A is having a central purchasing system. It has department B, C,D. And company A gets purchase order in XML format from B, C, D.

Now Company A needs to collect all Purchase orders and it has to store it in ORACLE8i database. And from that it has to create another "Request for proposal" for it's preferred vendors in XML.I am writing queries to insert or update into the Database. Tell me what are all the components I need to install in ORACLE8i.

Answer

Assuming you are using Java to implement you need the XML Parser and XML SQL Utility. If you are using a Java-based front-end to generate the purchase orders 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.

Portability: Using Parsers from Different Vendors?

Question

I am currently investigating SAX. I understand that both the Oracle and IBM parsers use DOM and SAX from W3.

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

Answer

Not if you stick to SAX interfaces and / or DOM interfaces for your implementation. That's what the standard interfaces are in place to assist you with.

Browsers that Support XML

Question

Is there a list of browsers that support XML?

Answer

The following browsers support the display of XML:

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

XML Support for Oracle 8.0.x

Question

I have a customer who is currently architecting some of their future systems to run on XML based interfaces. The customer is a large Wall Street Institution. However their current systems are all running 8.0.6, and they would like to have some of their XML concepts implemented on the existing systems due to high demand.

The customer would like to know if there is currently or in the future any plans to support XML based code within the database or if there are any adapters / cartridges that they 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 just fine outside the database against 8.0.6. It's just that they won't be able to:

- Run XML components inside the database
- Use interMedia *XML* Searching

which are both Oracle8*i*-only features.

EDI and XML

Question

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

Answer

Here are some thoughts on the subject:

- EDI is a difficult technology: EDI allows machine-to-machine communication in a format that developers cannot really read and understand.
- EDI messages are very difficult to debug. XML documents are readable and easier to apprehend.
- High training costs for EDI developers.

- 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 to map for example two different formats of PO#s.
- EDI is expensive: Expensive in training (see above) and in deployment too where EDI requires very powerful servers and imposes high requirements on specialized network (EDI runs on VANs and VAN 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. We are going to see a coexistence of these two - at least for a while. Large companies with an existing investment in EDI won't switch. They are probably going to use XML as a way to extend their existing EDI-based implementation, which raises the new question of XML / EDI integration.

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

What Oracle Tools Support B2B Exchanges?

Question

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

Answer

Oracle participates in several B2B standard bodies:

- 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 extract in and out XML-based messages from our e-business suite.

In general, Oracle Internet Platform as a whole provides an integrated and solid platform for B2B exchanges.

Oracle's Direction Regarding XML?

Question

What is Oracle's direction regarding XML?

Answer

Oracle's XML strategy is to enable using XML in ways which exploit all of the benefits of Oracle's current technology stack. Today you can combine Oracle XML components with the Oracle8*i* database and Advanced Queueing (AQ) to achieve some degree of conflict resolution, transaction verification, and so on. Oracle is working to make future Oracle8*i* releases more seamless with regard to conflict resolution, transaction verification, distributed 2 Phase Commit transactions,....

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

For future releases, Oracle plans to support sending XML payloads using AQs. This involves making XML queriable from SQL. This is being implemented.

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

XML Query

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

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

XML and BLOB (inside XML message)

Question

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

Answer

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

Maximum CLOB Size?

Question

If we store XML files as CLOBs in the Oracle8*i* database, what is the maximum file size?

Answer

2 Gigabytes. See the "Oracle8i Application Developer's Guide - Large Objects (LOBs)" at http://technet.oracle.com/doc/server.815/a68004/toc.htm for lots more info on LOB's and CLOBs as well as http://technet.oracle.com/tech/java/sqlj_jdbc/index2.htm?Code&files/advanced/advanced.htm for sample code.

Oracle 7.3.4: Data Transfers to Other Vendors Using XML

Question

My company has 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* in order to do so. Is there any

way of leveraging version 7 to do XML? I'm sure we'll move up to ver 8 sometime in the future but I don't know if we will within our timeline (next 3-4 months) for the next phase of the project I'm working on.

Answer

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

For JDBC drivers, please take a look at http://technet.oracle.com/tech/java/sqlj_jdbc/ Take a look at: Oracle 7 JDBC OCI and JDBC Thin Drivers

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

Question

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

Answer

You need the following:

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

The first three can be obtained from Oracle.

The fourth from SUN.

If you want to do this from a browser, you'll also need the following:

- A Java compliant web server
- XSQL Servlet

Building an XML Application: Software Needed?

Question

I have a CGI-PERL-Oracle7 application on Solaris 2.6 and I want to convert it to XML/XSL-JAVA-Oracle. I know most parts of the technologies, for example, SGML,

XML, JAVA etc., but I don't know how to start it in Oracle. What software I need from Oracle?

1. Can I use Apache instead of Oracle's 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 were created by my programs?
4. What should be between the WWW server and Oracle DB server? XSQL Servlet? Parser? JAVA VM? EJB? CORBA? SQLJ? JDBC? Oracle packages such as UTL_HTTP?

Answer

1. Yes you can. The Apache web server must now interact with Oracle through JDBC or other means. See the XSQL servlet. This is a servlet that can run on any servlet-enabled web server. This runs on Apache and connects to the database through a JDBC driver to the Oracle database.:
2. How far can I go with Oracle 7.3? You can go a long way. The only problem would be that you cannot run any of the java programs inside the server. i.e. you cannot load all the XML tools into the server. But you can connect to the database by downloading the Oracle JDBC utility for ORacle 7 and run all the programs as client-side utilities.:
3. Do I still need an XML Parser if all XML were created by my: programs? That depends on what you intend to do with the XML generated. If all your task is just to generate XML and send it out then you might not need it. But if you wanted to generate an XML DOM tree then you would need the parser. Also you would need it if you have incoming XML documents and you want to parse and store them somewhere. See the XML SQL utility for some help on this front.:
4. What should be between the WWW server and Oracle DB server? As explained before in question 1) you would need to have a servlet (or CGI) which interacts to Oracle through OCI or JDBC

Standard DTDs to Use for Orders, Shipment,...

Question

We have implemented Oracle8i and the XDK. Where can we find just basic, standard DTDs to build on for Orders, Shipments, and Acknowledgements?

Answer

A good place to start would be http://xml.org which is being set up for this purpose.

DTD to Database Schema

Question

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

Answer

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

Schema Map to XML

Question

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

1. Is there a best way to design tables and generate XML? (flat tables or objects/collections)
2. Can I use XML SQL Utilities in Pro*C? 3. What is limited size for generating XML doc. out from database? (If I can use Pro*C to call XSU)

Answer

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

XML in the Database: Performance

Question

I would like to know if there is a whitepaper which 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 / benchmark for XML products.

Faster Record Retrievals

Question

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(*) doesn't consider index, it takes nearly 20-30 seconds for the retrieved list to be seen on the browser window, if I just remove that count(*), the retrieval is quite fast, but then there is no previous and next as I had linked them to count(*).

Answer

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

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

Translating From Other Formats to XML

Question

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

Answer

For HTML, you can use utilities like Tidy or JTidy to turn HTML into well-formed HTML that can be transformed using XSLT.

For random text formats, you can try utilities like XFlat at http://www.unidex.com/xflat.htm. I saw a presentation on XFlat at XML99 and it seemed to be good but I haven't tried it myself.

XML File Size Limitations

Question

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

Answer

There are no XML limitations to an XML File size.

Maximum Size XML Document?

Question

1. Is there a maximum size for an XML document to provide data for PL/SQL (or SQL) across tables, provided that no CLOB are used?
2. The maximum size of XML document generated from Oracle8*i* to an XML document?

Answer

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

Generating Database Schema From a Rational Rose Tool

Question

It is possible to generate database schema in Oracle8*i* via a script with CREATE TABLE..., from an XML file generated by a Rational Rose design tool?

Answer

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

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

Further References

Other XML FAQs

Here are some other XML FAQ sites of interest:

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

Recommended XML/XSL Books

Question

Can you, please, recommend a good XML/XSL book?

Answer

A publisher group by the name of WROX has a number of helpful books, one of which is titled, "XML Design and Implementation" by Paul Spencer covers XML, XSL and development pretty well.

Comment

Although I do not have this book, my impression that it is good one on XML and XSL: The XML Bible. I read the updated chapter 14 from:

http://metalab.unc.edu/xml/books/bible/

and it gave me a good understanding of XSLT. Downloading this chapter is free so you can get a good impression.

Part II

XML-SQL Utility (XSU): Storing and Retrieving XML From the Database

Part II of this manual focuses on storing XML data in, and retrieving XML data from the Oracle*8i* database, and how to use XML-SQL Utility (XSU), to do these tasks.

It contains the following chapter:

4

Using XML-SQL Utility (XSU)

This chapter contains the following sections:

Accessing XML-SQL Utility

XML-SQL Utility (XSU) is provided with Oracle8i and it is made up of three files:

- $ORACLE_HOME/rdbms/jlib/xsu12.jar -- contains all the java classes which make up the XSU. The xsu12 requires JDK1.2.x or and JDBC2.x. This is the version of the xsu loaded into the database itself.
- $ORACLE_HOME/rdbms/jlib/xsu111.jar -- contains the same classes as xsu12.jar except that xsu111 requires JDK1.1.x and JDBC1.x.. JDK1.1.8 is the official JDK version supported by Oracle on the client side.
- $ORACLE_HOME/rdbms/admin/dbmsxsu.sql -- this is the SQL script which builds the XSU's PL/SQL API. xsu12.jar need to be loaded into the database before dbmsxsu.sql is executed. This script is automatically

By default the Oracle8i installer will install the XSU on your hard drive (in the locations specified above) as well as load it into the database. In the case that during initial installation you asked the installer not install the XSU, you can always run the installer later and have it install just the XSU and its dependent components. In this case the XSU will not be automatically loaded into the database; instead, you will have to take the following steps:

- If you haven't yet loaded the xmlparser for java into the database, go to $ORACLE_HOME/xdk/lib. Here you will find xmlparserv2.jar which you will need to load into the database. For how to accomplish this see "Loading JAVA Classes" in the "Oracle8i Java Stored Procedures Developer's Guide"
- Next go to $ORACLE_HOME/admin and execute the catxsu.sql script.

Note that the Oracle XML SQL Utility (XSU) is also available on OTN site: http://technet.oracle.com/tech/xml. You can check here for XSU updates.

Using XML-SQL Utility (XSU)

XML has rapidly become the format for data interchange. Today, a substantial amount of business data resides in object-relational databases. It is therefore necessary to transform this "relational" data to XML for purposes of communication. XML-SQL Utility (XSU) provides a simple way of achieving this data transformation, by mapping canonically any SQL query result to XML and vice versa.

For example, to retrieve the results of the employee table in scott's schema (which is available in all databases as a default schema), we can supply the query,

```
select * from scott.emp
```

to the utility and with the default settings will result in an XML document as shown below:-

```
<?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 can take a similar document and also insert it back in to the same table. XSU also provides APIs for updating and deleting XML documents.

When to Use XML-SQL Utility (XSU)

The XML-SQL Utility (XSU) provides the basic functionality to get and put data to and from the database. The utility provides a canonical mapping back and forth with some simple transformations (such as changing the name of the ROW tag generated for each row). Any complex transformations can be achieved by applying an industry standard XSL (XML Stylesheet Language) transformation over the document. Oracle XML Parser V2 supports a powerful XSL processor, XSLT, to perform transformations. XSL transformations can be also registered with the XSU directly, thus the XML generated by the XSU will be automatically transformed.

Note: If the primary focus of your application is to generate XML pages to web sources, then you can consider using the XSQL servlet. This is a standard Java servlet that can automate this process and provide a simpler XML template-like language to specify the transformations.

Where Can You Run XML-SQL Utility (XSU)?

The XML-SQL Utility is written in Java and can be run in any tier that supports Java, such as the following:

- In the database
- In Middle Tier Application servers which support Java -- call XSU from middle tier servers
- In Web Servers -- call the XSU Java APIs from within servlets running inside web servers.
- On the client -- use the XSU's Java API in your Java applications or use XSU's command line front end.

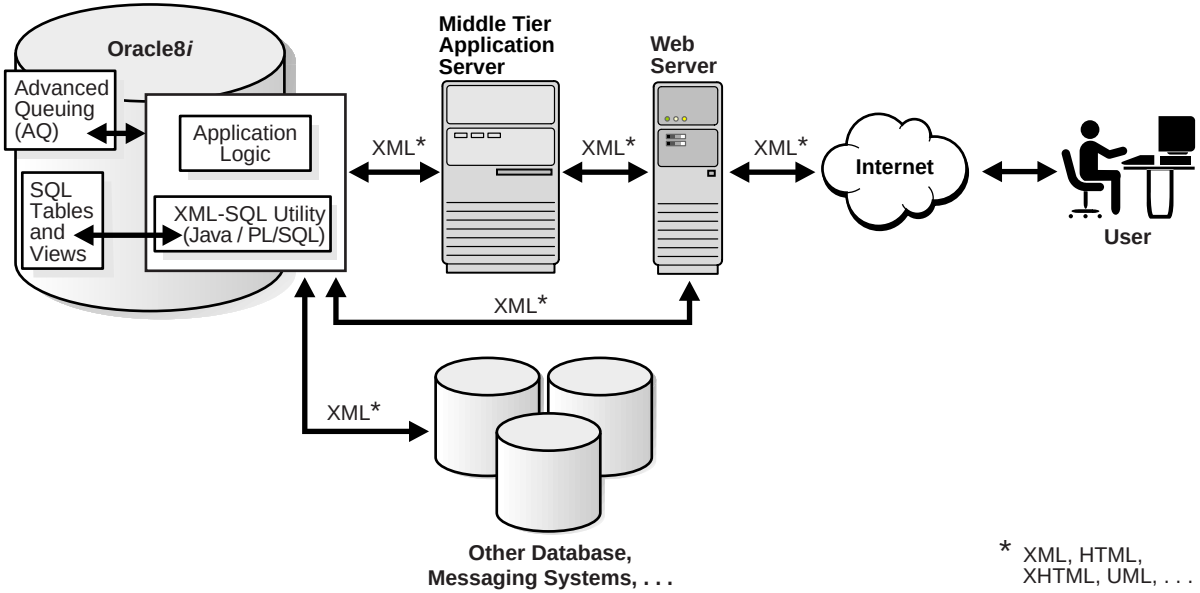
Running XML-SQL Utility in the Database

The java classes which make up the XSU can be loaded into the Oracle*8i* JServer; furthermore, the XSU contains a PL/SQL wrapper which published the XSU's Java API to PL/SQL creating a PL/SQL API. This way one can write new java applications which run inside the database and which directly access the XSU's Java API; one can write PL/SQL applicatons which access XSU through its PL/SQL API; or one can access the XSU's funtionality directly through SQL. Note that to load and run Java code inside the database you need a java enabled Oracle8i Sever.

Figure 4–1 shows how a typical architecture for such a system. XML generated from XSU running inside 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 inside the database or shipped outside via web servers or application servers.

Note in the figure that all lines are bi-directional. Since XSU can *generate* as well as *put* data, data can come from various sources to XSU running inside the database and can be put back in the appropriate database tables.

Figure 4–1 Running XML-SQL Utility in the Database



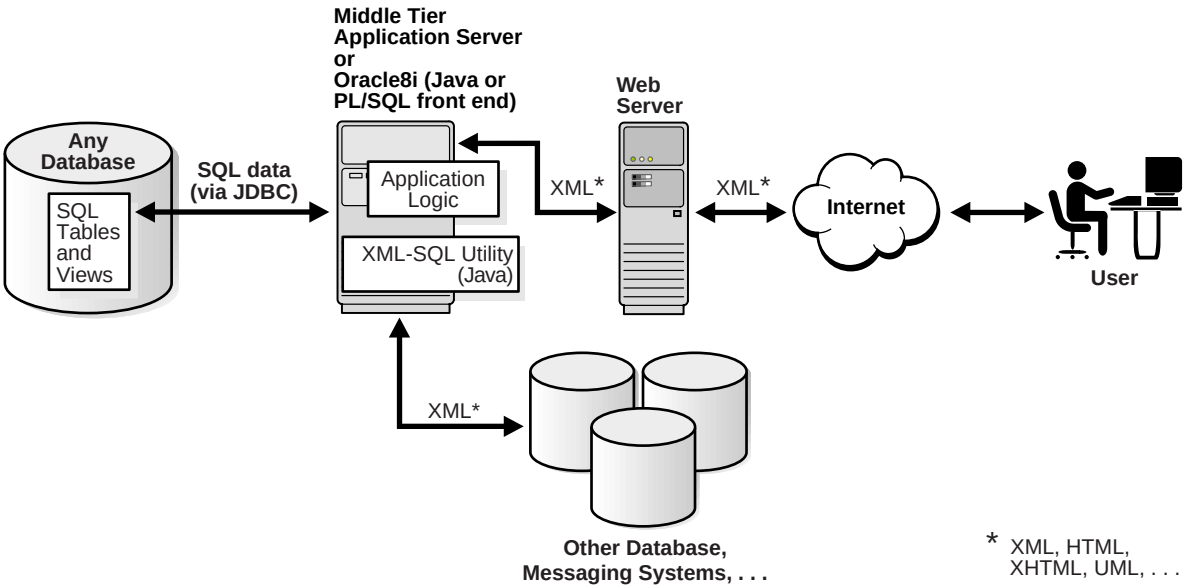
Running XML-SQL Utility in the Middle Tier

Your application architecture might force the use of an 'application server' in the middle tier that is separate from the database. This application tier could be an Oracle database, an Oracle 'application server', or a third party application server that supports Java programs.

You may want to generate XML in the middle tier from SQL queries or ResultSets for various reasons. For example, to integrate different JDBC data sources in the middle tier. In this case, you can use the Java version of XSU and then call the XSU Java API directly from Java programs running in the middle tier.

Figure 4–2, shows how a typical architecture for running XSU in a middle tier. Data from JDBC sources is converted by XSU in the middle tier 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*8i* database itself is used as the application server, then you can also use the PL/SQL front-end instead of Java.

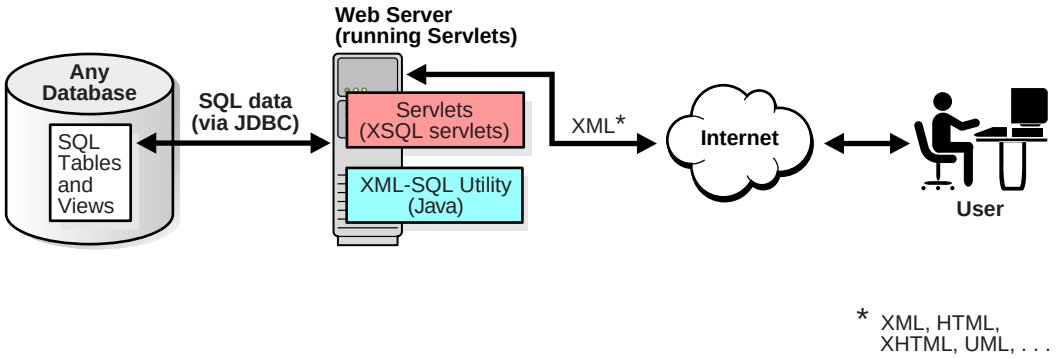
Figure 4–2 Running XML-SQL Utility in the MIddle Tier



Running XML-SQL Utility in a Web Server

The Java version of XSU can also be run in the web server itself, as long as the web server supports Java servlets. XSU can be directly called by Java servlets and the XML can be generated and put back from the web server itself. The XSQL servlet is a standard servlet provided by Oracle which calls the utility to generate and save XML data. If this is the primary intent for using this product, then consider using the XSQL servlet, as it eliminates a lot of servlet coding and provides a simpler template approach for XML processing from the web server.

> **See:** Chapter 19, "Using XSQL Servlet" for information about using the XSQL Servlet.



What Does XML-SQL Utility Work With?

XML-SQL Utility needs the following components in order to function:

- *A Data Source.* XML-SQL Utility (XSU) needs the JDBC drivers. The utility can work with any JDBC drivers but is optimized for Oracle's JDBC drivers. Oracle, does not make any guarantees or provide support for the XSU running against non-Oracle databases.
- *XML Parser.* XSU needs the Oracle XML Parser, Version 2. This is provided as part of the Oracle8i install, and is also available as part of the XSU install downloadable from Oracle Technology Network (OTN) web site.

XSU Features

XSU consists of a command line front end, a Java API and a PL/SQL API. The main features of the XSU are as follows:

- Supports generation of XML documents from any SQL query. It virtually supports all the datatypes supported in the Oracle8i database server.
- Supports dynamic generation of DTDs (Document Type Definitions). In the future we will support XMLSchemas as well.
- Supports simple transformations in the generation such as modifying the default tag names for the ROW element etc. One can also register a XSL transformation which is then applied to the generated XML documents on the fly.
- Can generate XML documents in their string or DOM representations.

- Supports insertion of XML into database tables / views. It also can update records or delete records from a database object, given an XML document.
- Complex nested XML documents can be easily generated and stored in to relational tables by creating object views over these flat tables and querying over these views. Object views can create structured data from existing relational data using Oracle*8i*'s object-relational infrastructure.

XSU Usage Guidelines

This section describes the basic usage steps for the various flavors of XSU, including the client side command line, Java APIs, and PL/SQL APIs. A comprehensive example and usage is described in the next section under usage patterns. The usage topics covered include the following:

- Mapping Primer
  - Mapping: Generating XML from SQL
  - Mapping: Storing SQL from XML - inserts, updates, deletes
- Using the XSU Command Line Front End
- Using the XSU Java API
- Using the XSU PL/SQL API -- dbms_xmlquery & dbms_xmlsave

Mapping Primer

Before explaining the steps involved in the usage of the various APIs, you need to know what the default mapping is from SQL to XML (Generating XML) and back (Storing XML). As explained earlier, XML-SQL Utility (XSU) provides a canonical mapping from SQL data to XML data and back. The following mappings are explained below:

- Mapping: Generating XML from SQL
- Mapping: Storing SQL from XML

Mapping: Generating XML from SQL

Given a SQL query, a canonical mapping is made to create an XML document. Consider a table, *emp* in the scott schema which has the structure,

```
CREATE TABLE emp
(
   EMPNO NUMBER,
   ENAME VARCHAR2(20),
   JOB VARCHAR2(20),
   MGR  NUMBER,
   HIREDATE DATE,
   SAL NUMBER,
   DEPTNO NUMBER
);
```

To convert the table elements to XML, we can fire off a query through one of the APIs available from the utility,

```
select * from scott.emp;
```

On executing this query, the utility will generate an XML document which contains a ROWSET tag to enclose the results of all the rows. Each ROW is encapsulated within a ROW tag. The ROW tag also contains an attribute "num" which identifies the row number for each element. Each scalar element maps to an XML element. Column names become the tag names for the element. In the present case, any column which cannot be a valid XML identifier name (such as empno$ or empno#) has to be changed to a valid XML name by supplying an alias in the select query:

```
<?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>
```

The default mapping can be changed by using the various options through the APIs. Scalar values map to flat XML documents.

Oracle*8i* supports the notion of object types, collections and object references. These provide structural modelling within the server. The mapping to XML for these preserve the structure. For example, consider department table which contains a department address structure and a list of employees.

The AddressType is an object type which defines the structure of an address object.

```
CREATE TYPE AddressType AS OBJECT (
   STREET VARCHAR2(20),
   CITY   VARCHAR2(20),
   STATE  CHAR(2),
   ZIP    VARCHAR2(10)
);
/
```

An employee type is also present which defines the structure of an employee. Note how the employee's address is defined using the address type.

```
CREATE TYPE EmployeeType AS OBJECT
(
  EMPNO NUMBER,
  ENAME VARCHAR2(20),
  SALARY NUMBER,
  EMPADDR AddressType
);
/
```

Now, we can create a list of employees by defining a list type.

```
CREATE TYPE EmployeeListType AS TABLE OF EmployeeType;
/
```

And finally, the department table is created with a department address and the list of employees. Each row of this table contains a nested collection of employee objects each of which contains the employee's descriptions including their name, salary and address.

```
CREATE TABLE Dept
(
  DEPTNO NUMBER,
  DEPTNAME VARCHAR2(20),
  DEPTADDR AddressType,
  EMPLIST  EmployeeListType
);
```

Assuming that valid values are stored in the department table, we can map the results of a select query on the table into an XML document shown below:-

```
<?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>
      <EMPLOYEE_TYPE 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>
        </EMPLOYEE_TYPE>
         <!-- additional employee types within the employee list -->
      </EMPLIST>
    </ROW>
    <!-- additional rows ... -->
  </ROWSET>
```

You can see from the example above, how the object type attributes map to nested elements in the XML document and collection types map to XML lists. The same nesting can also be achieved by using CURSOR subqueries. With the use of object views you can realize the same structures from existing relational tables.

Mapping: Storing SQL from XML

The XML-SQL utility provides the ability to map the XML documents to table rows and also provides the ability to update top level columns and to delete rows. The storage uses a simple mapping to map the element tag names to columns. XML strings are converted to the appropriate data types through default mappings. If the XML element is structured, you can map it to a SQL object type.

Inserts

For example, the *DEPTADDR* element would get mapped to the *AddressType* SQL type when inserting into the *Dept* table. When mapping to object types and collections, only the structure and the names need to match. So one can generate the XML from a column of the *AddressType2* and map it to a column of type *AddressType* provided the structure of the *Address* element is the same for both the types and the SQL type attribute names also match.

The insert case is handled simply by firing an insert statement and binding all the values of the elements in the VALUES clause of the insert statement. We would map the contents of each ROW element as a separate set of values to be inserted. So, if you take the XML document shown in the previous section, and ask XSU to insert it into the *Dept* table, the XSU will generate an insert statement of the form:

```
INSERT INTO Dept (DEPTNO, DEPTNAME, DEPTADDR, EMPLIST) VALUES (?,?,?,?)
```

and bind the values ,

```
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'),...)
```

If there is more than one ROW element in the XML document, then for each ROW, the XSU bind the values and executes the statement. The insert processing can be optimized to insert in batches, commit in batches all of which is explained in the "Insert Processing" on page 4-35.

Updates

Updates and deletes differ from insert in that they can affect more than one row in the database table. In the case of insert, each ROW element of the XML document can affect at most one row in the table, provided that there are no triggers or constraints on the table. However, in the case of updates and deletes, the XML element might match more than one row if the matching columns are not key columns in the table.

In case of updates, the user is expected to provide a list of key columns which the XSU will use 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 fire off the following update statement:

```
UPDATE DEPT SET DEPTNAME = ? WHERE DEPTNO = ?
```

and bind the values ,

```
DEPTNO <- 100
DEPTNAME <- SportsDept
```

In the update case, you can also opt to update only a set of columns and not all the elements present in the XML document.

Deletes

In the case of deletes, the user can opt 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 will try to match all the columns given in the document. Given a document such as:

```
<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, the utility will fire off a delete statement (one per ROW element) which would look 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')
```

Exact usage of all of these is explained in the "Basic Usage and advanced usage" sections. (**** cross reference this properly)

Using the XSU Command Line Front End

XSU comes with a simple command line front-end which gives user a quick access to XSU's XML generation and XML insertion functionalities. At this point, the XSU front end does not publish the update and delete functionalities of the XSU.

The command line options are provided through the java class `OracleXML`. Invoke it by calling:

```
java OracleXML
```

The above call will result the front-end usage information to be printed.

To be able to run the XSU front-end, you first need to specify where is the executable located. To do this add the XSU java library (xsu12.jar or xsu111.jar) to your CLASSPATH.

Now, since the XSU has a dependency on the Oracle XML Parser and the JDBC drivers, for the XSU to run, you need to make the location of these components also known. To do this, your CLASSPATH needs to include the locations of the Oracle XML Parser java library (xmlparserv2.jar) and the JDBC library (classes12.jar if using xsu12.jar or classes111.jar if using xsu111.jar).

Generating XML using XSU's Front-End

To use the generation capabilities, call XSU with the *getXML* parameter. For example, to generate an XML document by querying the *emp* table under *scott* schema,

```
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

getXML supports a wide range of options which are explained in the following section.

OracleXML - getXML Options

- -user "<username>/<password>"

  Used to specify 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>"

  Used to specify 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.

- -rowsetTag "<tag_name>"

  Used to specify rowset tag (i.e. 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>"

  Used to specify the row tag (the tag used to enclose the data coresponding 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>"

  Used to name 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 (i.e. "") as the row id attribute will tell the XSU to omit the attribute.

- -rowIdColumn "<row Id column name>"

  Used to specify that the value of one of the scalar columns from the query should be used as the value of the row id attribute.

- -collectionIdAttr "<collection id attribute name>"

  Used to name the attribute of a XML list element keeping track of the cardinality of the elements of the list (note: the generated XML lists correspond to either a cursor query, or collection). Specifying an empty string (i.e. "") as the row id attribute will tell the XSU to omit the attribute..

- -useNullAttrId

  Used to tell the XSU to use the attribute "NULL (TRUE/FALSE)" to indicate the nullness of an element.

- -styleSheet "<stylesheet URI>"

  Used to specify the stylesheet in the XML PI (Processing Instruction).

- -stylesheetType "<stylesheet type>"

  Used to specify the stylesheet type in the XML PI (Processing Instruction).

- -errorTag "<error tag name>"

  Used to specify the error tag -- the tag to enclose error messages which are formated into XML.

- -raiseNoRowsException

  Used to tell the XSU to raise an exception if no rows are returned.

- -maxRows "<maximum number of rows>"

  Used to specify the maximum number of rows to be retreived and converted to XML.

- -skipRows "<number of rows to skip>"

  Used to specify the number of rows to be skipped.

- -encoding "<encoding name>"

  Used to specify the characterset encoding of the generated XML.

- -dateFormat "<date format>"

  Used to specify the date format for the date values in the XML document.

- -fileName "<SQL query fileName>" | <sql query>

  Used to specify the file name which contains the query or specify the query itself.

Inserting XML using XSU's Front End

To put an XML document in to the *emp* table under *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's Front End putXML currently only publishes XSU's insert functionality and may be expanded in the future to also publish XSU's update and delete functionality.

OracleXML - putXML Options

The following lists the putXML options:

- -user "<username>/<password>"

  Used to specify 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>"

  Used to specify the JDBC database connect string. By default the connect string is: "jdbc:oracle:oci8:@"):

- -batchSize "<batching size>"

  Used to specify 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>"

  Used to specify 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>"

  Used to specify the row tag (the tag used to enclose the data coresponding 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>"

  Used to specify the date format for the date values in the XML document.

- -ignoreCase

  Used to make the matching of the column names with tag names case insensitive (e.g. "EmpNo" will match with "EMPNO" if ignoreCase is on).

- -fileName "<file name>" | -URL "<url>" | -xmlDoc "<xml document>"

  Used to specify 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 inlines the XML document as a string on the command line.

- <tableName>

  The name of the table to put the values into.

XML-SQL Utility for Java

The following two XSU classes are useful for the Java API programmer:

- `oracle.xml.sql.query.OracleXMLQuery`: Generates a XML document given a SQL query.
- `oracle.xml.sql.dml.OracleXMLSave`: Puts a XML document into tables and views

Generating XML

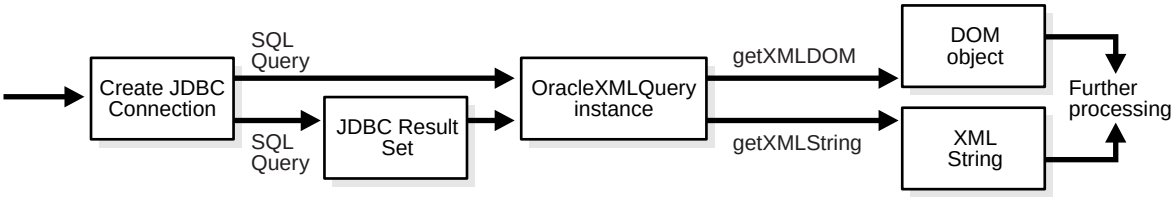
The `OracleXMLQuery` class makes up the XML generation part of the XSU's Java API.

Figure 4–3 illustrates the basic steps in the usage of OracleXMLQuery.

Perform these steps when generating XML:

1. First create a connection
2. Create an `OracleXMLQuery` instance by supplying a SQL string or a `ResultSet` object
3. Get the result as either a DOM tree or as an XML string

Figure 4–3 Generating XML With XML-SQL Utility for Java: Basic Steps

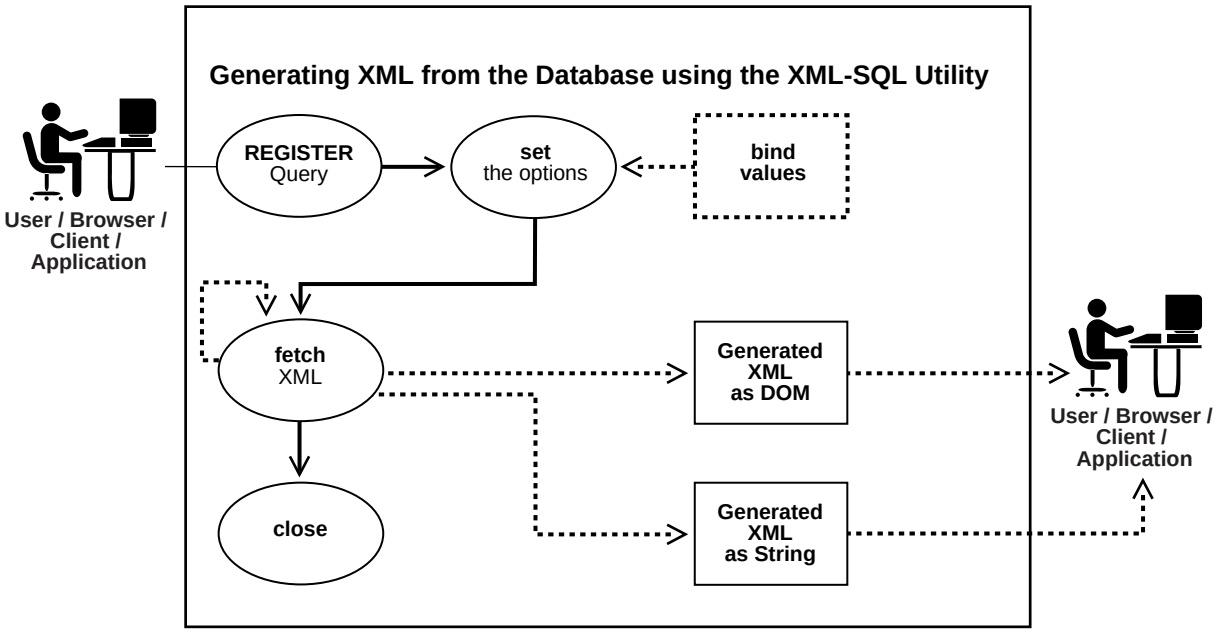


The following example, shows how a simple XML document can be generated.

XSU: Basic Generation of XML From SQL Queries

These examples illustrate how using the XSU you can get a XML document in its DOM or string representation given a SQL query. See Figure 4–4.

Figure 4–4 Generating XML With the XML-SQL Utility: Process and Options



XSU Example 1: Generating a String From emp table

The first step before getting the XML is to create a connection to the database. The connection can be obtained by supplying the JDBC connect string. You have to first *register* the Oracle JDBC class and then create the connection.

```
// 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 the OCI8 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: *Oracle8i Java Developer's Guide* for more details on this

Connecting With the Thin Driver

Here's an example of connecting using the thin driver.

```
// Create the connection.
Connection conn =
      DriverManager.getConnection("jdbc:oracle:thin:@dlsun489:1521:ORCL",
                                  "scott","tiger");
```

The thin driver requires the specification of 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

In the case of writing server side Java code, i.e., code that will run inside 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 that case you call the *defaultConnection()* on the *oracle.jdbc.driver.OracleDriver()* class to get the current connection.

```
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 rest of the notes will either assume the OCI8 connection from the client or that you already have a connection object created. Use the appropriate connection creation based on your needs.

Creating an OracleXMLQuery Class Instance

Once you have registered your connection, create an OracleXMLQuery class instance by supplying a SQL query to execute,

```
// 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.

XML String Output: You can get a XML string for the result by:

```
String xmlString = qry.getXMLString();
```

DOM Object Output: If, instead of a string, you wanted a DOM object instead, you can simply ask for a DOM output,

```
org.w3c.DOM.Document domDoc = qry.getXMLDOM();
```

and use the DOM traversals.

Here's a complete listing of the program to extract the XML string. This program gets the string and prints it out to the 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. Eexecute 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's JDeveloper to compile and run this program.

When run, this program prints out the XML file to the screen.

XSU Example 2: Generating DOM From emp table (Java)

A DOM (Document Object Model) is a standard defined by the W3C committee which represents an XML document in a parsed-tree like form. Each XML entity becomes a DOM node. Thus XML elements, attributes become DOM nodes and their children become child nodes.

To generate a DOM tree from the XML generated by the utility, it is efficient to directly ask for a DOM Document from the utility, as it saves the overhead of creating a string representation of the 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 example is shown below to get the DOM tree. The example "walks" 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, the XML-SQL Utility (XSU) takes the ResultSet or the query and generates the whole document from all the rows of the query. For getting say, 100 rows at a time, the user 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 say the first 5 rows of the query and then generate the Result. For getting these desired results, use XSU's skipRows and maxRows settings.

The skipRows parameter when set will force the generation to skip the desired number of rows before starting to generate the result. The maxRows on the other hand, would limit the number of rows that are converted to XML. If you set the skipRows to a value of 5 and maxRows to a value of 10, then the utility would skip the first 5 rows, and then generate the XML for the next 10 rows.

Keeping the Object Open

In web scenarios, you might want to keep the query object open for the duration of the user's session. For example, take 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, ask the utility to convert 10 rows at a time and to keep the ResultSet state alive, so that the next time we ask it for more results, it will start generating from the place the last generation finished.

When the Number of Rows or Columns in a Row Are Too Large

There is also the case that the number of rows, or the number of columns in a row may be 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 using the keepObjectOpen functionality.

keepObjectOpen Function

Typically, as soon as all results are generated, OracleXMLQuery internally closes the ResultSet, if it created one using the SQL query string given, since it assumes you no longer

want any more results. However, in the case described above, we need to maintain that state, so we need to call the keepObjectOpen function to keep the cursor alive.

XSU Example 3. Paginating Results: Generating an XML Page When Called (Java)

The following example, writes a simple class which maintains the state and generates the next page every 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

We saw how we can supply a SQL query and get the results as XML. In the last example, we saw how we can retrieve results in a paginated fashion. However in web cases, we might want to retrieve the previous page and not just the next page of results. To provide this scrollable functionality, we can use the Scrollable ResultSet. Use the ResultSet object to move back and forth within the result set and use the utility to generate the XML everytime.

XSU Example 4: Generating XML from JDBC ResultSets (Java)

We will show how to use the JDBC ResultSet and generate XML from that. Note that using the ResultSet might be necessary in cases which are not handled directly by the utility (for example, setting the batch size, binding values,...) We will extend the previously defined pageTest class so that we 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 Example 5: Generating XML from Procedure Return Values (REF CURSORS) (Java)

The OracleXMLQuery class provides XML conversion only for query string or for ResultSets. But in your application if you had PL/SQL procedures which returned REF cursors, how would you do the conversion?

In this case, you can use the above 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 which can be iterated upon to get a set of values. These REF cursors are converted in to `OracleResultSet` objects in the Java world.

You can execute these procedures, get the `OracleResultSet` object and then send that in to the OracleXMLQuery object to get the desired XML.

Take this PL/SQL function which 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;
/
```

Now, everytime this function is called, it opens a cursor object for the query, "select * from emp" and returns that cursor instance. If you wanted to convert this to XML, you can do the following:

```
import org.w3c.dom.*;
import oracle.xml.parser.v2.*;
import java.sql.*;
import oracle.jdbc.driver.*;
import oracle.xml.sql.query.OracleXMLQuery;
import java.io.*;
public class REFCURtest
{
   public static void main(String[] argv)
     throws SQLException
   {
      String str;
      Connection conn  = getConnection("scott","tiger"); // create connection

      // Create a ResultSet object by calling the PL/SQL function
      CallableStatement stmt =
         conn.prepareCall("begin ? := testRef.testRefCur(); end;");

      stmt.registerOutParameter(1,OracleTypes.CURSOR); // set the define type
```

```
      stmt.execute();   // Execute the statement.
      ResultSet rset = (ResultSet)stmt.getObject(1);  // Get the ResultSet

      OracleXMLQuery qry = new OracleXMLQuery(conn,rset); // prepare Query class
      qry.setRaiseNoRowsException(true);
      qry.setRaiseException(true);
      qry.keepCursorState(true);        // set options (keep the cursor alive..
      while ((str = qry.getXMLString())!= null)
           System.out.println(str);

      qry.close();    // close the query..!

      // Note since we supplied the statement and resultset, closing the
      // OracleXMLquery instance will not close these. We would need to
      // explicitly close this ourselves..!
      stmt.close();
      conn.close();
   }
    // Get the connection given the user name and password..!
    private static Connection getConnection(String user, String passwd)
      throws SQLException
    {
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
      Connection conn =
          DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
     return conn;
   }

}
```

To apply the stylesheet on the other hand, use the `applyStylesheet()` command. This forces the stylesheet to be applied before generating the output.

Raising No Rows Exception

When there are no rows to process the utility simply returns a null string. But it might be desirable to get an exception everytime 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 the utility raises a oracle.xml.sql.OracleXMLSQLNoRowsException. This is a run time exception and need not be caught unless needed. The following code extends the previous examples to use the Exception instead of checking for null strings.

XSU Example 6: No Rows Exception (Java)

```
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();
     }
   }
}
```

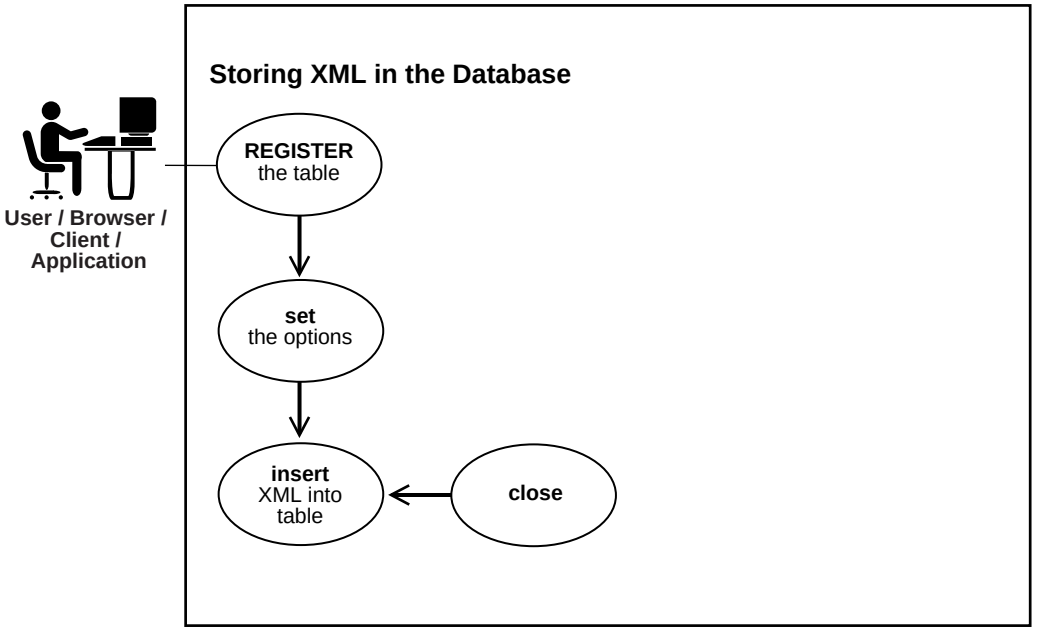
Note how the condition to check the termination changed from checking for the result to be null to an exception handler.

Storing XML

Now that we have seen how queries can be converted to XML, let see how we can put the XML back into the tables or views using the utility. The class oracle.xml.sql.dml.OracleXMLSave provides such functionality. It provides methods to insert the XML into tables, update existing tables with the XML document and to delete rows from the table based on the XML element values.

In all these cases the given XML document is parsed, the elements examined to match the tag names to those of the column names in the target table or view. The elements are then converted to the SQL types and then bound to the appropriate statement. The process and options for storing XML using the XSU are shown in Figure 4–5.

Figure 4–5 Storing XML in the Database Using the XML-SQL Utility: Process and Options



The document is assumed to contain a list of ROW elements each of which constitute a separate DML operation, namely, insert, update or delete on the table or view.

Insert Processing

The steps to insert a document into a table or view is to simply supply the table or the view name and then the document. The utility parses the document (if a string is given) and then creates an insert statement which it binds all the values into. By default, the utility 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 Example 7: Inserting XML Values into all Columns (Java)

This example inserts XML values into all columns:

```
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");
  // Assume that the user passes in this document. Save it in to the table.!
      sav.insertXML(argv[1]);
      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, MGR, SAL, DEPTNO) VALUES(?,?,?,?,?,?);
```

will be generated and the element tags in the input XML document matching the column names will be matched and their values bound. For the code snippet shown above, if we send it the 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>
```

we 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 have been taken as a null value.

XSU Example 8: Inserting XML Values into Only Certain Columns (Java)

In certain cases, you may not want to insert values into all columns. This may be true when the values that we are getting is not the complete set and we need triggers or default values to be used for the rest of the columns. The code below shows how this can be done.

Assume that we are getting the values only for the employee number, name and job and the salary, manager, deptno and hiredate field gets filled in automatically. First create a list of column names that we 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[1] = "EMPNO";
      colNames[2] = "ENAME";
      colNames[3] = "JOB";

      sav.setUpdateColumnList(colNames); // set the columns to update..!

      // Assume that the user passes in this document as the first argument!
      sav.insertXML(argv[1]);
      sav.close();
   }
   // Get the connection given the user name and password..!
    private static Connection getConnection(String user, String passwd)
      throws SQLException
    {
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
      Connection conn =
          DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
```

```
      return conn;
    }
}
```

An insert statement of the form,

```
insert into scott.emp (EMPNO, ENAME, JOB) VALUES (?, ?, ?);
```

is generated. Note that in the above example, if the inserted document contains values for the other columns (JOB, HIREDATE etc.), those will be ignored.

Also an insert is performed for each ROW element that is present in the input. These inserts are batched by default.

Update Processing

Now that we know how to insert values into the table from XML documents, let us see how to update only certain values. If we 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>
```

we can call the update processing to update the values. In the case of update, we need to supply the utility with the list of key column names. These form part of the where clause in the update statement. In the emp table shown above, the employee number (EMPNO) column forms the key and we use that for updates.

XSU Example 9: Updating Using the keyColumns (Java)

This example updates the emp table 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[1] = "EMPNO";
      sav.setKeyColumnList(keyColNames);

      // Assume that the user passes in this document as the first argument!
      sav.updateXML(argv[1]);
      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 would be generated. For the first ROW element, we would generate an update statement to update the SAL and JOB fields as shown below:-

```
update scott.emp SET SAL = 1800 and DEPTNO = 30 WHERE EMPNO = 7369;
```

and for the second ROW element,

```
update scott.emp SET SAL = 2000 and HIREDATE = 12/31/1992 WHERE EMPNO = 2290;
```

XSU Example 10: Updating a Specified List of Columns (Java)

However, in a lot of cases we might 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 we can ignore other tags which occur in the document. Note that when we specify a list of columns to update, an element corresponding to one of the update columns, if absent, will be treated as NULL.

If we know that all the elements to be updated are the same for all the ROW elements in the XML document, then we 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[1] = "EMPNO";
      sav.setKeyColumnList(keyColNames);

      // we create the list of columns to update..!
      // Note that if we do not supply this, then for each ROW element in the
      // XML document, we 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[1] = "SAL";
      updateColNames[2] = "JOB";
      sav.setUpdateColumnList(updateColNames); // set the columns to update..!

      // Assume that the user passes in this document as the first argument!
      sav.updateXML(argv[1]);
      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

In the case of delete, you can set the list of key columns. These columns will be put as part of the where clause of the delete. 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 Example 11: Deleting Operations Per ROW (Java)

Consider the delete example shown below,

```
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[1]);
      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;
   }
}
```

If we use the same XML document shown for the update example, we 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 12: Deleting Specified Key Values (Java)

If we instead want the delete to only use the key values as predicates, we can use the `setKeyColNames` 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[1] = "EMPNO";
      sav.setKeyColumnList(keyColNames);

      // Assume that the user passes in this document as the first argument!
      sav.deleteXML(argv[1]);
      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 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.

Using the XML-SQL Utility for PL/SQL

The XML-SQL Utility PL/SQL API reflects the Java API in the generation and storage. The `DBMS_XMLQuery` and `DBMS_XMLSave` are the two packages that reflect the functions in the java classes - OracleXMLQuery and `OracleXMLSave`.

Both these packages have a context handle associated with them. Create a context by calling one of the constructor-like functions to get the handle and then use the handle in all subsequent calls.

Generating XML with DBMS_XMLQuery

Generating XML results in a CLOB that contains the XML document. The steps involved in using the generation engine follow:

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 something like, select * from emp where empno = :EMPNO_VAR. Here the user binds 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 etc.
4. Fetch the XML as a CLOB using the getXML() functions. The getXML can be called to generate the XML with or without a DTD.
5. Close the context.

Here are some examples that use this PL/SQL package.

XSU Example 13: Generating XML From Simple Queries (PL/SQL)

In this example, we will try to select rows from the emp table and get a XML document as a CLOB. We first get the context handle by passing in a query and then call the `getXMLClob` routine to get the CLOB value. The document will be in the same encoding as that of 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 Example 13a: Printing CLOB to Output Buffer

The `printClobOut()` is a simple procedure that prints the CLOB to the output buffer. If you run this PL/SQL code in SQL*Plus, you will see the result of the CLOB being printed out to screen. Set the `serveroutput` to on in order to see the results.

The `printClobOut` is shown below:-

```
/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 Example 14: Changing ROW and ROWSET Tag Names (PL/SQL)

The PL/SQL APIs also provide the ability to change the ROW and the ROWSET tag names. These are the default names that are put around each row of the result and around the whole document respectively. The procedures, `setRowTagName` and `setRowSetTagName` accomplish this as shown below:

```
--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 and each row separated using the EMP tag.

XSU Example 15: Paginating Results Using setMaxRows() and setSkipRows()

The results from the query generation can be paginated by using the `setMaxRows` and `setSkipRows` functions. The former 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. The skipRows parameter 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, we 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 the XML-SQL Utility Java API, call the *keepObjectOpen()* function to make sure that the state is maintained between fetches. The default behavior is to close the state after a fetch is done. In the case of multiple fetches, we need to figure out 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 PL/SQL API provides the ability to set the stylesheet header in the result XML or apply a stylesheet itself to the result XML document, before generation. The latter is a huge performance win since otherwise the XML document has to be generated as a CLOB, sent to the parser again and then the stylesheet applied. In this case, internally the utility generates a DOM document, calls the parser, applies the stylesheet and then generates the result. The procedure `setStylesheetHeader()` sets the stylesheet header in the result. This simply adds the XML processing instruction to include the stylesheet. The `useStyleSheet()` procedure on the other hand uses the stylesheet to generate the result.

Binding Values in XSU (PL/SQL)

The PL/SQL API provides the ability to bind values to the SQL statement. The SQL statement can contain named bind variables. The variables have to start with a ':' in front of them to signal that they are bind variables. The steps involved in using the bind variable is as follows,

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 which we 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, we will set the empno and ename values as shown below:-

   ```
   DBMS_XMLQuery.clearBindValues(queryCtx);
   DBMS_XMLQuery.setBindValue(queryCtx,'EMPNO',20);
   DBMS_XMLQuery.setBindValue(queryCtx,'ENAME','John');
   ```

3. Fetch the results. This will apply the bind values to the statement and then get the result corresponding to the predicate empno = 20 and ename = 'John'.

   ```
   DBMS_XMLQuery.getXMLClob(queryCtx);
   ```

4. Re-bind values if necessary, For example to change the ENAME alone to "scott" and re-execute the query,

   ```
   DBMS_XMLQuery.setBindValue(queryCtx,'ENAME','Scott');
   ```

   The rebinding of ENAME will now use Scott instead of John.

XSU Example 15a: Binding Values to the SQL Statement

The following example illustrates the use of bind variables in the SQL statement:

```
declare
  queryCtx DBMS_XMLquery.ctxType;
  result CLOB;
begin

queryCtx := DBMS_XMLQuery.newContext(
       'select * from emp where empno = :EMPNO and ename = :ENAME');

DBMS_XMLQuery.clearBindValues(queryCtx);
DBMS_XMLQuery.setBindValue(queryCtx,'EMPNO',7566);
DBMS_XMLQuery.setBindValue(queryCtx,'ENAME','JONES');

result := DBMS_XMLQuery.getXML(queryCtx);
```

```
--printClobOut(result);

DBMS_XMLQuery.setBindValue(queryCtx,'ENAME','Scott');

result := DBMS_XMLQuery.getXML(queryCtx);

--printClobOut(result);
end;
/
```

Storing XML in the Database Using DBMS_XMLSave

The steps involved in using the XML-SQL Utility storage engine follow:

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. In case of 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.

XSU Insert Processing in PL/SQL

The steps to insert a document into a table or view is to simply supply the table or the view name and then the document. The utility parses the document (if a string is given) and then creates an insert statement which it binds all the values into. By default, the utility 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 Example 16: Inserting Values into All Columns (PL/SQL)

This example creates a procedure, insProc, which takes in an XML document as a CLOB and a table name to put the document into and then inserts the document:

```
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');
```

will generate an insert statement of the form,

```
insert into scott.emp (EMPNO, ENAME, JOB, MGR, SAL, DEPTNO) VALUES(?,?,?,?,?,?);
```

and the element tags in the input XML document matching the column names will be matched and their values bound. For the code snippet shown above, if we send it the 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>
```

we 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 have been taken as a null value.

XSU Example 17: Inserting Values into Only Certain Columns (PL/SQL)

In certain cases, we may not want to insert values into all columns. This might be true when the values that we are getting is not the complete set and we need triggers or default values to be used for the rest of the columns. The code below shows how this can be done.

Assume that we are getting the values only for the employee number, name and job and the salary, manager, deptno and hiredate field gets filled in automatically. We create a list of column names that we 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 the `setUpdateColumnName()` procedure repeatedly, passing in a column name to update every time. The column name settings can be cleared using the `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 we call the procedure passing in a CLOB as a document, an insert statement of the form,

```
insert into scott.emp (EMPNO, ENAME, JOB) VALUES (?, ?, ?);
```

is generated. Note that in the above example, if the inserted document contains values for the other columns (JOB, HIREDATE etc.), those will be ignored.

Also an insert is performed for each ROW element that is present in the input. These inserts are batched by default.

Update Processing

Now that we know how to insert values into the table from XML documents, let us see how to update only certain values. If we 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>
```

we can call the update processing to update the values. In the case of update, we need to supply the utility with the list of key column names. These form part of the where clause in the update statement. In the emp table shown above, the employee number (EMPNO) column forms the key and we use that for updates.

XSU Example 18: Updating an XML Document Using keyColumns(PL/SQL)

,.....

```
create or replace procedure testUpdate ( xmlDoc IN clob) is
  updCtx DBMS_XMLSave.ctxType;
```

```
  rows number;
begin

   updCtx := DBMS_XMLSave.newContext('scott.emp');  -- get the context
   DBMS_XMLSave.clearUpdateColumnList(updCtx); -- clear the update settings..

   DBMS_XMLSave.setKeyColumn(updCtx,'EMPNO'); -- set EMPNO as key column
   rows := DBMS_XMLSave.updateXML(updCtx,xmlDoc);  -- update the table.
   DBMS_XMLSave.closeContext(updCtx);             -- close the context..!

end;
/
```

In this example, when the procedure is executed with a CLOB value that contains the document described above, two update statements would be generated. For the first ROW element, we would generate an update statement to update the SAL and JOB fields as shown below:-

```
update scott.emp SET SAL = 1800 and DEPTNO = 30 WHERE EMPNO = 7369;
```

and for the second ROW element,

```
update scott.emp SET SAL = 2000 and HIREDATE = 12/31/1992 WHERE EMPNO = 2290;
```

XSU Example 19: Specifying a List of Columns to Update (PL/SQL)

However, in a lot of cases we might 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 we can ignore other tags which occur in the document. Note that when we specify a list of columns to update, an element corresponding to one of the update columns, if absent, will be treated as NULL.

If we know that all the elements to be updated are the same for all the ROW elements in the XML document, then we 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

In the case of delete, you can set the list of key columns. These columns will be put as part of the where clause of the delete. 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 Example 20: Deleting Operations per ROW (PL/SQL)

Consider the delete example shown below,

```
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 we use the same XML document shown for the update example, we would end up with two delete statements,

```
DELETE FROM scott.emp WHERE empno=7369 and sal=1800 and deptno=30;
DELETE FROM scott.emp WHERE empno=2200 and sal=2000 and hiredate=12/31/1992;
```

The delete statements were formed based on the tag names present in each ROW element in the XML document.

XSU Example 21: Deleting by Specifying the Key Values (PL/SQL)

If we instead want the delete to only use the key values as predicates, we can use the `setKeyColNames` 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 Example 22: ReUsing the Context Handle (PL/SQL)

In all the three cases described above, insert, update and delete, the same context handle can be used to do more than one operation. i.e. one 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 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.ctxHandle := 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(xmlDoc);
  end;

  procedure update(xmlDoc in clob) is
  begin
    DBMS_XMLSave.updateXML(xmlDoc);
  end;

  procedure delete(xmlDoc in clob) is
  begin
    DBMS_XMLSave.deleteXML(xmlDoc);
  end;

  begin
    saveCtx := DBMS_XMLSave.getCtx('scott.emp'); -- create the context once..!
    DBMS_XMLSave.setKeyColumnName('EMPNO');      -- set the key column name.
  end;
end;
```

```
/
```

In the above package, we create a context once for the whole package (thus the session) and then reuse the same context for performing inserts, udpates and deletes. Note that the key column ('EMPNO') would be used both for updates and deletes as a way of identifying the row.

The users of this package can now call any of the three routines to update the *emp* table.

```
testDML.delete(xmlclob);
testDML.update(xmlclob);
```

All of these calls would use the same context. This would improve the performance of these operations, particularly if these operations are performed frequently.

Advanced Usage Techniques

Exception Handling in Java

OracleXMLSQLException class

The utility catches all exceptions that occur during processing and throws an `oracle.xml.sql.OracleXMLSQLException` which is a run time exception. The calling program thus does not have to catch this exception all the time. If the program can still catch this exception and do the appropriate action. The exception class provides functions to get the error message and also get the parent exception, if any. For example, the program shown below, catches the run time exception and then gets the parent exception.

OracleXMLNoRowsException class

This exception is generated when the `setRaiseNoRowsException` is set in the OracleXMLQuery class during generation. This is a subclass of the `OracleXMLSQLException` class and can be used as an indicator of the end of row processing during generation.

```
import java.sql.*;
import oracle.xml.sql.query.OracleXMLQuery;

public class testException
{
   public static void main(String argv[])
     throws SQLException
   {
      Connection conn = getConnection("scott","tiger");

      // wrong query this will generate an exception
      OracleXMLQuery qry = new OracleXMLQuery(conn, "select * from emp where sd
 = 322323");

      qry.setRaiseException(true); // ask it to raise exceptions..!

      try{
        String str = qry.getXMLString();
      }catch(oracle.xml.sql.OracleXMLSQLException e)
      {
        // Get the original exception
        Exception parent = e.getParentException();
```

```
        if (parent instanceof java.sql.SQLException)
        {
           // perform some other stuff. Here we 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;
   }
}
```

Exception Handling in PL/SQL

Here is a 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 (FAQs): XML-SQL Utility (XSU)

What Schema Structure to Use With XSU to Store XML?

Question

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 (i.e. has a nested structure), you should use an object-relational schema. The XML above will canonically map to such a schema. An appropriate db. schema would be the following:

```
create type address_type as object
 (
 street varchar2(40),
 city varchar2(20),
 state varchar2(10),
 zip varchar2(10)
 );
 /
 create type customer_type as object
 (
customerid number(10),
firstname varchar2(20),
```

```
lastname varchar2(20),
homeaddress address_type
 );
/
create table customer_tab ( customer customer_type);
```

In the case you wanted to load customer.xml via the XSU into a relational schema, you could still do it by creating objects in views on top of your relational schema.

For example, you would have a relational table which would contain all the information:

```
create table cust_tab
 ( customerid number(10),
   firstname varchar2(20),
   lastname varchar2(20),
   state varchar2(40),
   city varchar2(20),
   state varchar2(20),
   zip varchar2(20)
 );
```

Then you would create a customer view which contains a customer object on top of it, as in:

```
create view customer_view as
select customer_type(customerid, firstname, lastname,
address_type(state,street,city,zip))
from cust_tab;
```

Finally, you could flatten your XML using XSLT and then insert it directly into your relational schema. This is the least recommended option.

Storing XML Data Across Tables

Question

Can XML- SQL Utility store XML data across tables?

Answer

Currently XML-SQL Utility (XSU) can only store to a single table. It maps a canonical representation of an XML document into any table / view. But of course there is a way to store XML with the XSU across table. One can do this using XSLT

to transform any document into multiple documents and insert them separately. Another way is to define views over multiple tables (object views if needed) and then do the inserts ... into the view. If the view is inherently non-updatable (because of complex joins, ...), then one can use INSTEAD-OF triggers over the views to do the inserts.

Using XML-SQL Utility to Load XML Stored in Attributes

Question

I would like to use the XML-SQL Utility to load XML where some of the data is stored in attributes; yet, the XML-SQL Utility seems to ignore the XML attributes. What can I do?

Answer

Unfortunately, for now you will have to use XSLT to transform your XML document (i.e. change your attributes into elements). The XML-SQL Utility does assume canonical mapping from XML to a db. schema. This takes away a bit from the flexibility, forcing the user to sometimes resort to XSLT, but at the same time, in the common case, it doesn't burden the user with having to specify a mapping.

XML-SQL Utility is Case Sensitive: Use ignoreCase or...

Question

I am trying to insert the following XML document (dual.xml):

```
<ROWSET>
   <row>
      <DUMMY>X</DUMMY>
   </row>
</ROWSET>
```

Into the table "dual" using the command line front end of the XSU, like in:

```
java OracleXML putxml -filename dual.xml dual
```

and I get the following error:

```
oracle.xml.sql.OracleXMLSQLException: No rows to modify -- the row enclosing tag
missing.  Specify the correct row enclosing tag.
```

Answer

By default the XML SQL Utility is case sensitive, so it looks for the record separator tag which by default is "ROW"; yet, all it can find is "row". Another related common mistake is to case mismatch one of the element tags. For example if in dual.xml the tag "DUMMY" was actually "dummy" than the XML SQL Utility would also raise an error complaining that if couldn't find a matching column in the table "dual". So user has two options -- use the correct case or use the "ignoreCase" feature.

Generating Database Schema from a DTD

Question

Given a DTD, will the XML SQL Utility generate the database schema?

Answer

No. Due to a number of shortcomings of the DTD, this functionality is not available. Once XML Schema standard is finalized this functionality will become feasible.

Using XML-SQL Utility Command Line

Question

I am using the XML SQL Utility's command line front end, and I am passing a connect string but I get a TNS error back. Can you provide examples of a thin driver connect string and an OCI8 driver connect string?

Answer

An example of an JDBC thin driver connect string is: "jdbc:oracle:thin:<user>/<password>@<hostname>:<port number>:<DB SID>"; furthermore, the db. has to have a active TCP/IP listener. A valid OCI8 connect string would be: "jdbc:oracle:oci8:<user>/<password>@<hostname>".

Does XML-SQL Utility Commit After INSERT, DELETE, UPDATE?

Question

Does XML SQL Utility commit after it's done inserting/deleting/updating? What happens if an error occurs.

Answer

By default the XML SQL Utility executes a number of insert (or del 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.

By default the XML SQL Utility does no explicit commits. If the autocommit is on (default for the JDBC connection) then after each batch of statement executions a commit happens. The user can override this by turning autocommit off and then specifying after how many statement executions should a commit occur which can be done using the "`setCommitBatch`" feature.

Finally, what happens if an error occurs... Well, the XSU rolls back to either the state the target table was before the particular call to the XSU, or the state right after the last commit made during the current call to the XSU.

Part III

Managing Content and Documents with XML

Part III of this manual describes how to use *inter*Media Text to enhance and turbocharge the search and retrieval of your XML database application. *inter*Media Text can also be used to search and retrieve from non-database applications.

It includes an *inter*Media Text case study as well as several case studies illustrating XML-based content management.

Part III contains the following chapters:

5

Using *inter*Media Text to Search and Retrieve Data from XML Documents

This chapter contains the following sections:

- Frequently Asked Questions (FAQs): interMedia Text

Introducing *inter*Media Text

This chapter describes the following aspects of *inter*Media Text:

- Creating an Index
- Building an XML Query Application with *inter*Media Text

Note: *inter*Media Text is strictly a server-based implementation.

See Also: http://technet.oracle.com/products/intermedia/

Overview of *inter*Media Text

*inter*Media Text can be used to search XML documents. It extends Oracle*8i* by indexing any text or document stored in Oracle*8i*. It can also search documents in the operating system (flat files) and URLs.

*inter*Media Text enables the following:

- Content-based queries, such as, finding text and documents which contain particular words, using familiar, standard SQL.
- File-based text applications to use Oracle8i to manage text and documents in an integrated fashion with traditional relational information.
- Concept searching of English language documents
- Theme analysis of English language documents using the CTX_DOC PL/SQL package
- Highlighting hit words. With interMedia Text, you can render a document in different ways. For example, you can present documents with query terms highlighted, either the "words" of a word query or the "themes" of an ABOUT query in English. Use the CTX_DOC.MARKUP or HIGHLIGHT procedures for this.
- With *inter*Media Text PL/SQL packages for document presentation and thesaurus maintenance

*inter*Media Text is packaged with the other *inter*Media products, namely, image, audio, video, and geographic location services for web content management applications.

Users can query XML data stored in the database directly, without using *inter*Media Text. However, *Inter*media Text is useful for boosting query performance.

> See: *Oracle8i interMedia Text Reference* for more information about *inter*Media Text.

Installing *inter*Media Text

*inter*Media, including *inter*Media Text, is a standard feature that comes with every Oracle8*i* Standard, Enterprise, and Personal edition license. However, it needs to be selected during installation. No special installation instructions are required.

*inter*Media Text is essentially a set of schema objects owned by CTXSYS. These objects are linked to the Oracle kernel. The schema objects are present when you perform an Oracle8*i* installation.

*inter*Media Text Users and Roles

With *inter*Media Text you can use CTXSYS user to administer users and CTXAPP role to create and delete *inter*Media Text *preferences* and use *inter*Media Text PL/SQL packages:

CTXSYS User

This user is created at install time. Administer *inter*Media Text users as this user. It has the following privileges:

- Modify system-defined preferences
- Drop and modify other user preferences
- Call procedures in the CTX_ADM PL/SQL package to start servers and set system-parameters
- Start a ctxsrv server
- Query all system-defined views
- Perform all the tasks of a user with the CTXAPP role

CTXAPP Role

Any user can create an *inter*Media Text index and issue a Text query. For additional tasks, use the CTXAPP role. This is a system-defined role that allows users to perform the following tasks:

- Create and delete *inter*Media Text *preferences*
- Use *inter*Media Text PL/SQL packages, such as the CTX_DDL package

Querying with the CONTAINS Operator

*Inter*Media Text's main purpose is to provide an implementation for the CONTAINS operator. The CONTAINS operator is used in the WHERE clause of a SELECT statement to specify the query expression for a Text query.

See Also: "Building Query Applications with interMedia Text".

CONTAINS Syntax

Here is the CONTAINS syntax:

```
CONTAINS(
        [schema.]column,
        text_query       VARCHAR2,
        [label            NUMBER])
RETURN NUMBER;
```

where:

Table 5–1 CONTAINS Operator: Syntax Description

Syntax	Description
[schema.]column	Specify the text column to be searched on. This column must have a Text index associated with it.
text_query	Specify the query expression that defines your search in column.
label	Optionally specify the label that identifies the score generated by the CONTAINS operator.
Returns	For each row selected, CONTAINS returns a number between 0 and 100 that indicates how relevant the document row is to the query. The number 0 means that Oracle found no matches in the row. You can obtain this score with the SCORE operator. Note: You must use the SCORE operator with a label to obtain this number.

CONTAINS Example 1

The following example illustrates how the CONTAINS operator is used in a SELECT statement:

```
SELECT id FROM my_table
   WHERE
   CONTAINS (my_column, 'receipts') > 0
```

The 'receipts' parameter of the CONTAINS function is called the "Text Query Expression". See "Text Query Expression" for an example of how to use this.

Note: The SQL statement with the CONTAINS function requires a text index in order to run.

CONTAINS Example 2: Using Score Operator with a Label

The following example searches for all documents in the text column that contain the word Oracle. The score for each row is selected with the SCORE operator using a label of 1:

```
SELECT SCORE(1), title from newsindex
          WHERE CONTAINS(text, 'oracle', 1) > 0;
```

The CONTAINS operator must always be followed by the > 0 syntax which specifies that the score value calculated by the CONTAINS operator must be greater than zero for the row to be selected.

CONTAINS Example 3: Using the SCORE Operator

When the SCORE operator is called, such as in a SELECT clause, the operator must reference the label value as in the following example.

```
SELECT SCORE(1), title from newsindex
          WHERE CONTAINS(text, 'oracle', 1) > 0 ORDER BY SCORE(1) DESC;
```

Assumptions Made in this Chapter's Examples

XML text is aVARCHAR2 or CLOB type in an Oracle8*i* database table with character semantics.

*inter*Media Text can also deal with documents on a file system or on URLs. We do not consider these document types here.

To simplify the examples included in this chapter we consider a subset of the *inter*Media Text options.

We have made the following assumptions here:

- All XML data here is represented using US-ASCII, a 7 bit character set.
- Issues about whether a character such as "*" is treated as white space or as part of a word are not included.
- Storage characteristics of the Oracle schema object that implement the TEXT index are not considered.
- We focus here on the SECTION GROUP parameter in the CREATE INDEX or ALTER INDEX statement[1]. Here is an example of using SECTION GROUP in CREATE INDEX:

```
CREATE INDEX my_index
  ON my_table ( my_column )
  INDEXTYPE IS ctxsys.context
  PARAMETERS ( 'SECTION GROUP my_section_group' ) ;
```

- Specifically, we focus on using AUTO_SECTION_GROUP and XML_SECTION_GROUP.
- Tagged or marked up data. We focus here on how to handle XML data. interMedia Text handles many other kinds of data.

[1] Other parameter types are available for CREATE INDEX and ALTER INDEX. These are DATASTORE, FILTER, LEXER, STOPLIST, and WORDLIST.

Using interMedia Text to Search XML Documents

To search and retrieve data from XML documents you need to do the following:

- Create an interMedia Text Index
- Build a query application

These procedures are described in the following sections.

*inter*Media Text Indexes

To create an index for *inter*Media Text follow these steps:

1. Determine the role you need to use and GRANT ctxapp privilege. See "interMedia Text Users and Roles".
2. Set up the tables and data, if not already available
3. Create an *inter*Media Text index to use CONTAINS in the query
4. Create a preference for parameterization of the *inter*Media Text index
5. Parameterize the *inter*Media Text index using the CTX_DDL package's Create_Section_Group and Add_Field_Section procedures.

Using the CTX_DDL PL/SQL Package

The CTX_DDL PL/SQL supplied package creates and manages the objects required for interMedia Text indexes. The detailed list of CTX_DDL package's stored procedures and functions can be found in *Oracle8i interMedia Text Reference.* Use the CTXAPP role when executing the CTX_DDL package.

The following CTX_DDL procedures are used in this chapter:

- create_preference
- create_section_group
- add_attr_section
- add_field_section
- add_special_section
- add_zone_section

Listing the Required Roles for Each CTX Package

If you are uncertain which role to use with the CTX packages, run the following script. This lists the roles required for each CTX package:

```
connect ctxsys/ctxsys
column "Package"                          format a15
column "Role needed to execute Package" format a30

select dba_objects.object_name  "Package",
       dba_tab_privs.grantee    "Role needed to execute Package"
  from dba_objects, dba_tab_privs
  where dba_objects.object_name = dba_tab_privs.table_name (+)
    and dba_objects.object_type in ( 'PACKAGE', 'PROCEDURE', 'FUNCTION' )
    and dba_objects.object_name like 'CTX_%'
    order by "Role needed to execute Package";
```

This results in the following output:

```
Package         Role needed to execute Package
--------------- ------------------------------
Ctx_Ddl         CTXAPP
Ctx_Output      CTXAPP
Ctx_Thes        CTXAPP
Ctx_Contains    PUBLIC
Ctx_Query       PUBLIC
```

```
Ctx_Doc         PUBLIC
Ctx_Adm
```

CTX_DDL Procedures

The following lists the CTX_DDL procedures used in this chapter, along with each procedure's arguments and descriptions.

```
procedure create_preference
 argument name                  type                    in/out default?
 ------------------------------ ----------------------- ------ --------
 preference_name                varchar2                in
 object_name                    varchar2                in

procedure create_section_group
 argument name                  type                    in/out default?
 ------------------------------ ----------------------- ------ --------
 group_name                     varchar2                in
 group_type                     varchar2                in

procedure add_attr_section
 argument name                  type                    in/out default?
 ------------------------------ ----------------------- ------ --------
 group_name                     varchar2                in
 section_name                   varchar2                in
 tag                            varchar2                in

procedure add_field_section
 argument name                  type                    in/out default?
 ------------------------------ ----------------------- ------ --------
 group_name                     varchar2                in
 section_name                   varchar2                in
 tag                            varchar2                in
 visible                        boolean                 in     default

procedure add_special_section
 argument name                  type                    in/out default?
 ------------------------------ ----------------------- ------ --------
 group_name                     varchar2                in
 section_name                   varchar2                in

procedure add_zone_section
 argument name                  type                    in/out default?
 ------------------------------ ----------------------- ------ --------
```

group_name	varchar2	in
section_name	varchar2	in
tag	varchar2	in

XML_SECTION_GROUP Attribute Sections

In Oracle8*i* versions higher than 8.1.5, the XML section group offers the ability to index and search within *attribute* values. Consider a document with the following lines:

```
<comment author="jeeves">
  I really like interMedia Text
</comment>
```

Now XML section group offers an attribute section. This allows the inclusion of attribute values to index. For example:

```
ctx_ddl.add_attr_section('mysg','author','comment@author');
```

The syntax is similar to other add_section calls. The first argument is the name of the section group, the second is the name of the section, and the third is the tag, in the form <tag_name>@<attribute_name>. This tells interMedia Text to index the contents of the author attribute of the comment tag as the section "author".

Query syntax is just like for any other section:

```
WHERE CONTAINS ( ... ,'jeeves WITHIN author...',...)...
```

and finds the document.

Attribute Value Sensitive Section Search

Attribute sections allow you to search the contents of attributes. They do not allow you to use attribute values to specify sections to search. For instance, given the document:

```
<comment author="jeeves">
  I really like interMedia Text
</comment>
```

You can find this document by asking:

```
jeeves within comment@author
```

which is equivalent to "find me all documents which have a comment element whose author attribute's value includes the word jeeves".

However, there is no way to ask for something like:

```
interMedia within comment where (@author = "jeeves")
```

in other words, "find me all documents where interMedia appears in a comment element whose author is jeeves". This feature -- attribute value sensitive section searching -- is planned for future versions of the product.

Dynamic Add Section

Because the section group is defined before creating the index, 8.1.5 is limited in its ability to cope with changing structured document sets; if your documents start coming with new tags, or you start getting new doctypes, you have to re-create the index to start making use of those tags.

8.1.6 and higher allows you to add new sections to an existing index without rebuilding the index, using alter index and the new add section parameters string syntax:

```
add zone  section <section_name> tag <tag>
add field section <section_name> tag <tag> [ visible | invisible ]
```

For instance, to add a new zone section named tsec using the tag title:

```
alter index <indexname> rebuild
parameters ('add zone section tsec tag title')
```

To add a new field section named asec using the tag author:

```
alter index <indexname> rebuild
parameters ('add field section asec tag author')
```

This field section would be invisible by default, just like when using add_field_section. To add it as visible field section:

```
alter index <indexname> rebuild
parameters ('add field section asec tag author visible')
```

Dynamic add section only modifies the index meta-data, and does not rebuild the index in any way. This means that these sections take effect for any document indexed after the operation, and do not affect any existing documents -- if the index already has documents with these sections, they must be manually marked for re-indexing (usually with an update of the indexed column to itself).

This operation does not support addition of special sections. Those would require all documents to be re-indexed, anyway. This operation cannot be done using rebuild online, but it should be a fairly quick operation.

AUTO_SECTION_GROUP

Use AUTO_SECTION_GROUP group type to automatically create a zone section for each start-tag/end-tag pair in an XML document. The section names derived from XML tags are case-sensitive as in XML.

Attribute sections are created automatically for XML tags that have attributes. Attribute sections are named in the form attribute@tag.

Stop sections, empty tags, processing instructions, and comments are not indexed.

The following limitations apply to automatic section groups:

- You cannot add zone, field or special sections to an automatic section group.
- Automatic sectioning does not index XML document types (root elements.) However, you can define stop-sections with document type.
- The length of the indexed tags including prefix and namespace cannot exceed 64 characters.
- Tags longer than this are not indexed.

Automatic Sectioning in XML Documents

The following command creates a section group called auto with the AUTO_SECTION_GROUP group type. This section group automatically creates sections from tags in XML documents.

```
begin
ctx_ddl_create_section_group('auto', 'AUTO_SECTION_GROUP');
end;
create index myindex on docs(htmlfile) indextype is ctxsys.context
parameters('filter ctxsys.null_filter section group auto');
```

Creating an *inter*Media Text Index

The following example illustrates how to create an *inter*Media Text index. The example is presented in tutorial fashion. Creating an interMedia Text index involves the following steps:

- 1 Determine the Role you Need and GRANT ctxapp Privilege
- 2 Set up Data, if Not Already Available
- 3 Creating an interMedia Text Index in Order to use CONTAINS
- 4 Creating a Preference: You Need to Express the Parameterization with a "Preference"
- 5 Parameterizing the Preference

1 Determine the Role you Need and GRANT ctxapp Privilege

Determine the role you need. See "Listing the Required Roles for Each CTX Package" and grant the appropriate privilege.

```
CONNECT system/manager
GRANT ctxapp to scott;
CONNECT scott/tiger
```

2 Set up Data, if Not Already Available

The following example creates a table, my_table, and inserts data into the table.

```
-- Set up some data

CREATE TABLE my_table (id number(5) primary key, my_column clob );

INSERT INTO my_table VALUES ( 1,
  '<author>Fred Smith</author>' ||
  '<document>I had a nice weekend in the mountains.</document>' );
INSERT INTO my_table VALUES ( 2,
  '<author>Jack Jones</author>' ||
  '<document>My month at the coast was relaxing.</document>' );
INSERT INTO my_table VALUES ( 3,
  '<publisher>Dog House</publisher>' ||
  '<document>His year in Provence was fulfilling.</document>' ||
  '<junk>banana</junk>' );
COMMIT;
```

3 Creating an *inter*Media Text Index in Order to use CONTAINS

The following example shows that you need a Text index in order to use CONTAINS.

3.1 You Need an *inter*Media Text Index in Order to Use CONTAINS

This example shows that you need an *inter*Media Text index in order to use "CONTAINS":

```
SELECT my_column FROM my_table
  WHERE CONTAINS
    ( my_column, 'smith WITHIN author'
    ) > 0;
```

The following error message is obtained:

```
DRG-10599: column is not indexed
```

3.2 Creating a Default Parameterized *inter*Media Text Index

This example shows that a default parameterized *inter*Media Text index does not support XML functionality:

```
CREATE INDEX my_index ON my_table ( my_column )
  INDEXTYPE IS Ctxsys.Context /* implies defaults */;

SELECT my_column FROM my_table
  WHERE CONTAINS
    ( my_column, 'smith WITHIN author'
    ) > 0;
```

The following error message is obtained:

```
DRG-10837: section author does not exist
```

See "5 Parameterizing the Preference".

4 Creating a Preference: You Need to Express the Parameterization with a "Preference"

Consider a preference, "my_section_group". It must be created before we can refer to it! This example shows that you need to express parameterization with a preference:

```
DROP INDEX my_index;

CREATE INDEX my_index ON my_table ( my_column )
  INDEXTYPE IS Ctxsys.Context
  PARAMETERS ( 'SECTION GROUP my_section_group' );
```

The following error message is obtained:

```
DRG-12203: section group my_section_group does not exist
```

5 Parameterizing the Preference

You need to do more than just CREATE 'my_section_group', you need to parameterize it too. Use CTX_DDL.Create_Section_Group

5.1 Parametrizing the Preference, my_section_group

This example shows the need to parameterize the my_section_group preference:

```
DROP INDEX my_index;

BEGIN
  Ctx_Ddl.Create_Section_Group
    ( group_name => 'my_section_group',
      group_type => 'xml_section_group'
    );
end;
/
CREATE INDEX my_index ON my_table ( my_column )
  INDEXTYPE IS Ctxsys.Context
  PARAMETERS ( 'SECTION GROUP my_section_group' );

SELECT my_column FROM my_table
  WHERE CONTAINS
    (my_column, 'smith WITHIN author'
    ) > 0;
```

The following error message is obtained:

```
DRG-10837: section author does not exist
```

5.2 Parameterize the Preference, 'my_section_group', Correctly Using CTX_DDL.Create_Section_Group and CTX_DDL.Add_Field_Section

This example shows that after creating the *inter*Media Text index and the preference, my_section_group, you need to parameterize the preference correctly.

- First use the CTX_DDL.create_section_group to create the section group.
- Then use CTX_DDL.Add_Field_Section for every tag used after WITHIN in your query statement.

> **See Also:** Using Table CTX_OBJECTS and CTX_OBJECT_ATTRIBUTES View for a brief description of the CTX_DDL package, and *Oracle8i interMedia Text Reference* for detailed notes on CTX_DDL.:

*inter*Media Text Example 1: Creating an Index — Creating a Preference and Correctly Parameterizing it

```
DROP INDEX my_index;
BEGIN
Ctx_Ddl.Drop_Section_Group
  ( group_name => 'my_section_group'
  );
END;
/

BEGIN
Ctx_Ddl.Create_Section_Group /* We're dealing here with XML, not say HTML */
  (  group_name => 'my_section_group',
     group_type => 'xml_section_group'
      );

Ctx_Ddl.Add_Field_Section /* THIS IS KEY */
  (  group_name   =>'my_section_group',
     section_name =>'author',/* do this for EVERY tag used after "WITHIN" */
     tag          =>'author'
  );

 Ctx_Ddl.Add_Field_Section /* THIS IS KEY */
      (  group_name   =>'my_section_group',
         section_name =>'document',/*do this for EVERY tag after "WITHIN" */
         tag          =>'document'
      );

  ...
END;
/

CREATE INDEX my_index ON my_table ( my_column )
```

```
  INDEXTYPE IS Ctxsys.Context
  PARAMETERS ( 'SECTION GROUP my_section_group' );

SELECT my_column FROM my_table
  WHERE CONTAINS
    ( my_column, 'smith WITHIN author'
    ) > 0;
```

The last example is correct.

*inter*Media Text Example 2: Creating a Section Group with AUTO_SECTION_GROUP

The following command creates a section group called autogroup with the AUTO_SECTION_GROUP group type. This section group *automatically* creates sections from tags in XML documents.

```
BEGIN
ctx_ddl.create_section_group('autogroup', 'AUTO_SECTION_GROUP');
END;
```

To index your documents you can use the following statement:

```
CREATE INDEX myindex ON docs(htmlfile) INDEXTYPE IS ctxsys.context
   parameters('filter ctxsys.null_filter section group autogroup');
```

> **Note:** You can add attribute sections only to XML section groups. When you use AUTO_SECTION_GROUP, attribute sections are created automatically. Attribute sections created automatically are named in the form tag@attribute.

*inter*Media Text Example 3: Creating a Section Group with XML_SECTION_GROUP

The following command creates a section group called xmlgroup with the XML_SECTION_GROUP group type.

```
BEGINE
ctx_ddl.create_section_group('xmlgroup', 'XML_SECTION_GROUP');
END;
```

You can add sections to this group using `CTX_DDL.ADD_SECTION`.

To index your documents, you can use the following statement:

```
CREATE INDEX myindex ON docs(htmlfile) INDEXTYPE IS ctxsys.context
```

```
parameters('filter ctxsys.null_filter section group xmlgroup');
```

Further Examples for Creating Section Group Indexes?

Further examples for creating section group indexes are at the following site:

http:/ / technet.oracle.com

Then select the following:

- Internet Servers. This is under Products on the side navigation bar.
- interMedia. This is under Oracle8i towards the top of the page.
- Training. This is on the top navigation bar.
- Oracle8i interMedia Text 8.1.5/8.1.6 - Training or similar.

Building Query Applications with *inter*Media Text

Building query applications with *inter*Media Text includes the following topics:

- Text Query Expression
- Querying with Attribute Sections
- Querying SECTION GROUPS
- Procedure for Building a Query Application with interMedia Text
- Creating Sections in XML Documents that are Document Type Sensitive

Text Query Expression

Text Query Expression allows you to do the following:

- Express all word-based criteria that are standard in the text retrieval industry, such as, boolean word combinations, proximity, and phrases searches.
- Perform theme based search criteria, for example, (...CONTAINS (my_ column,'About (customizing XML presentations)')...). Here, the phrase, 'customizing XML presentations' typically, does not occur in the retrieved document.

> **See Also:** *Oracle8i interMedia Text Reference* for more information on Text Query Expression.

*inter*Media Text Example 4: Using Text Query Expressions

This example shows the set up and usage of the text query expression in a SELECT statement:

- Creating a Thesaurus
- Creating Table christie
- Creating Table ctx_mutab

Creating a Thesaurus

```
-- run as ctxsys
begin
  Ctx_Thes.Drop_Thesaurus ( 'default' );
exception
  when others then
    /* not an error if...
```

```
          DRG-11701: thesaurus default does not exist */
    if instr ( SQLERRM, 'DRG-11701' ) != 0
    then
      null;
    else
      raise_application_error ( -20000, SQLERRM );
    end if;
end;
/
begin
  Ctx_Thes.Create_Thesaurus (
    name     => 'default',
    casesens => false );

  Ctx_Thes.Create_Phrase (
    tname   => 'default',
    phrase  => 'crime' );
  Ctx_Thes.Create_Phrase (
    tname   => 'default',
    phrase  => 'murder',
    rel     => 'NT',
    relname => 'crime' );
  Ctx_Thes.Create_Phrase (
    tname   => 'default',
    phrase  => 'death',
    rel     => 'RT',
    relname => 'murder' );
  Ctx_Thes.Create_Phrase (
    tname   => 'default',
    phrase  => 'kill',
    rel     => 'RT',
    relname => 'murder' );
  Ctx_Thes.Create_Phrase (
    tname   => 'default',
    phrase  => 'strangling',
    rel     => 'NT',
    relname => 'murder' );
  Ctx_Thes.Create_Phrase (
    tname   => 'default',
    phrase  => 'thirteen' );
  Ctx_Thes.Create_Phrase (
    tname   => 'default',
    phrase  => '13',
    rel     => 'SYN',
    relname => 'thirteen' );
```

```
end;
/
```

Creating Table christie

```
Set Define Off
begin
  execute immediate
    'drop table christie';
exception
  when others then
    /* not an error if...
          ORA-00942: table or view does not exist */
    if instr ( SQLERRM, 'ORA-00942' ) != 0
    then
      null;
    else
      raise_application_error ( -20000, SQLERRM );
    end if;
end;
/

create table christie ( id number, title varchar2(700) );
insert into christie ( title ) values ( '<T>Thirteen At Dinner</T> - <A>Agatha
Christie</A>' );
insert into christie ( title ) values ( '<T>The 4:50 from Paddington</T> -
<A>Agatha Christie</A>' );
insert into christie ( title ) values ( '<T>Blue Geranium</T> - <A>Agatha
Christie</A>' );
insert into christie ( title ) values ( '<T>The fiction of Agatha Christie</T> -
<A>John Smith</A>' );
insert into christie ( title ) values ( '<T>Caribbean with quite a few
intervening words between it and Mystery</T> - <A>Agatha Christie</A>' );
commit;
update christie set id = rownum;
commit;
alter table christie
  add constraint christie_pk primary key ( id )
  using index;
create unique index christie_title on christie ( title );
drop index christie_title;

begin
  Ctx_Ddl.Drop_Preference ( 'my_basic_lexer' );
exception
```

```
  when others then
    /* not an error if...
          preference does not exist */
    if instr ( SQLERRM, 'DRG-10700' ) != 0
    then
      null;
    else
      raise_application_error ( -20000, SQLERRM );
    end if;
end;
/
begin
  Ctx_Ddl.Drop_Section_Group ( 'my_basic_section_group' );
exception
  when others then
    /* not an error if...
          section group does not exist */
    if instr ( SQLERRM, 'DRG-12203' ) != 0
    then
      null;
    else
      raise_application_error ( -20000, SQLERRM );
    end if;
end;
/
begin
  Ctx_Ddl.Create_Preference ( 'my_basic_lexer', 'basic_lexer'           );
  Ctx_Ddl.Set_Attribute     ( 'my_basic_lexer', 'index_themes', 'false' );
  Ctx_Ddl.Set_Attribute     ( 'my_basic_lexer', 'index_text',   'true'  );
  Ctx_Ddl.Create_Section_Group
    (
       group_name => 'my_basic_section_group',
       group_type => 'basic_section_group'
    );
  Ctx_Ddl.Add_Field_Section
    (
      group_name   => 'my_basic_section_group',
      section_name => 'title',
      tag          => 't',
      visible      => true
    );
  Ctx_Ddl.Add_Field_Section
    (
      group_name   => 'my_basic_section_group',
      section_name => 'author',
```

```
      tag          => 'a',
      visible      => true
    );
end;
/
create index christie_title on christie ( title )
  indextype is ctxsys.context
  parameters ( 'lexer my_basic_lexer section group my_basic_section_group' );
begin
  Ctx_Ddl.Drop_Preference ( 'my_basic_lexer' );
  Ctx_Ddl.Drop_Section_Group ( 'my_basic_section_group' );
end;
/
```

Creating Table ctx_mutab

```
Set Define Off
begin
  execute immediate
    'drop table ctx_mutab';
exception
  when others then
    /* not an error if...
          ORA-00942: table or view does not exist */
    if instr ( SQLERRM, 'ORA-00942' ) != 0
    then
      null;
    else
      raise_application_error ( -20000, SQLERRM );
    end if;
end;
/
create table ctx_mutab
  (
    query_id number constraint ctx_mutab_pk primary key,
    document clob
  );

begin
  execute immediate
    'drop sequence ctx_mutab_seq';
exception
  when others then
    /* not an error if...
```

```
          ORA-02289: sequence does not exist */
    if instr ( SQLERRM, 'ORA-02289' ) != 0
    then
      null;
    else
      raise_application_error ( -20000, SQLERRM );
    end if;
end;
/
create sequence ctx_mutab_seq start with 1;
```

Accepting Text Query Expression and Running the Query

This accepts a Text Query Expression, runs the query, and for each hit displays the output of Ctx_Doc.Markup.

```
Set Define Off
create or replace procedure Qry_And_Markup
  (
    p_qry in varchar2 default null
  )
is
  v_query_id   number;
  v_document   clob;
  v_amount     number;
  v_nof_hits   integer := 0;
begin
  if p_qry is not null
  then
    for j in
      (
        select score(0) s, id from christie
          where contains ( title, p_qry, 0 ) > 0
          order by s desc
      )
    loop
      select ctx_mutab_seq.nextval
        into v_query_id
        from dual;
      Ctx_Doc.Markup
        (
          index_name   => 'christie_title',
          textkey      => to_char ( j.id ),
          text_query   => p_qry,
          restab       => 'ctx_mutab',
```

```
        query_id     => v_query_id,
        starttag     => Show.Start_Tag,
        endtag       => Show.End_Tag
      );

    select document
      into v_document
      from ctx_mutab
      where query_id = v_query_id;

    v_amount := 4000;

    Show.Table_Row
      (
        p_cell_1 => to_char ( j.s ),
        p_cell_2 =>
          Dbms_Lob.Substr
            (
              lob_loc    => v_document,
              amount     => v_amount,
              offset     => 1
            )
      );
    v_nof_hits := v_nof_hits + 1;
  end loop;

  if v_nof_hits < 1
  then
    Show.Table_Row
      (
        p_cell_1 => 'No hits'
      );
  end if;
else
  for j in
    (
      select title from christie
    )
  loop
    Show.Table_Row
      (
        p_cell_1 => j.title
      );
  end loop;
end if;
```

```
end Qry_And_Markup;
/
Show Errors
```

Querying with Attribute Sections

You can query within attribute sections when you index with either XML_SECTION_GROUP or AUTOMATIC_SECTION_GROUP as your section group type.

Assume you have an XML document as follows:

```
<book title="Tale of Two Cities">It was the best of times.</book>
```

You can define the section title@book to be the attribute section title. You can do so with the CTX_DLL.ADD_ATTR_SECTION procedure or dynamically after indexing with ALTER INDEX.

Note: When you use the AUTO_SECTION_GROUP to index XML documents, the system automatically creates attribute sections and names them in the form attribute@tag.

If you use the XML_SECTION_GROUP, you can name attribute sections anything with CTX_DDL.ADD_ATTR_SECTION.

To search on Tale within the attribute section title, issue the following query:

```
'Tale WITHIN title'
```

Constraints for Querying Attribute Sections

The following constraints apply to querying within attribute sections:

- Regular queries on attribute text do not hit the document unless qualified in a within clause. Assume you have an XML document as follows:

  ```
  <book title="Tale of Two Cities">It was the best of times.</book>
  ```

  A query on Tale by itself does not produce a hit on the document unless qualified with WITHIN title@book. This behavior is like field sections when you set the visible flag set to false.

- You cannot use attribute sections in a nested WITHIN query.

- Phrases ignore attribute text. For example, if the original document looked like:

  ```
  Now is the time for all good <word type="noun"> men </word> to come to the aid.
  ```

Then this document would hit on the regular query good men, ignoring the intervening attribute text.

WITHIN queries can distinguish repeated attribute sections. This behavior is like zone sections but unlike field sections. For example, for the following document:

```
<book title="Tale of Two Cities">It was the best of times.</book>
<book title="Of Human Bondage">The sky broke dull and gray.</book>
```

Assume the book is a zone section and book@author is an attribute section. Consider the query:

```
'(Tale and Bondage) WITHIN book@author'
```

This query does not hit the document, because tale and bondage are in different occurrences of the attribute section book@author.

Querying SECTION GROUPS

Distinguishing Tags Across DocTypes

In previous releases, the XML section group was unable to distinguish between tags in different DTD's. For instance, perhaps you have a DTD for storing contact information:

```
<!DOCTYPE contact>
<contact>
  <address>506 Blue Pool Road</address>
  <email>dudeman@radical.com</email>
</contact>
```

Appropriate sections might look like:

```
ctx_ddl.add_field_section('mysg','email',  'email');
ctx_ddl.add_field_section('mysg','address','address');
```

This is fine until you have a different kind of document in the same table:

```
<!DOCTYPE mail>
 <mail>
   <address>dudeman@radical.com</address>
</mail>
```

Now your address section, originally intended for street addresses, starts picking up email addresses, because of tag collision.

Specifying Doctype Limiters to Distinguish Between Tags

Oracle8*i* release 8.1.5 and higher allow you to *specify doctype limiters* to distinguish between these tags across doctypes. Simply specify the doctype in parentheses before the tag as follows:

```
ctx_ddl.add_field_section('mysg','email','email');
ctx_ddl.add_field_section('mysg','address','(contact)address');
ctx_ddl.add_field_section('mysg','email','(mail)address');
```

Now when the XML section group sees an address tag, it will index it as the address section when the document type is `contact`, or as the email section when the document type is `mail`.

Doctype-Limited and Unlimited Tags in a Section Group

If you have both doctype-limited and unlimited tags in a section group:

```
ctx_ddl.add_field_section('mysg','sec1','(type1)tag1');
ctx_ddl.add_field_section('mysg','sec2','tag1');
```

Then the limited tag applies when in the doctype, and the unlimited tag applies in all other doctypes.

Querying is unaffected by this -- the query is done on the section name, not the tag, so querying for an email address would be done like:

```
radical WITHIN email
```

which, since we have mapped two different kinds of tags to the same section name, finds documents independent of which tags are used to express the email address.

Procedure for Building a Query Application with *inter*Media Text

To build the query application with *inter*Media Text carry out the following indexing steps first. These steps are described in a previous section, "interMedia Text Indexes".

1. Grant ctxapp permission to the user
2. Create a parameterized Text index.
3. Index the documents
4. Present the documents that satisfy the query

The next step is to build your query application. To do so follow these steps:

1. Create a preference using the procedure, CTX_DDL.create_preference. See "1 Create a Preference"
2. Set preference's attributes using CTX_DDL.Add_Attr_Section and so on. See "2 Set the Preference's Attributes".
3. Create your query syntax

You can query within attribute sections when you index with either XML_ SECTION_GROUP or AUTOMATIC_SECTION_GROUP as your section group type.

Note:

- Not everything in your document may be searchable. You must first state what is searchable using the.....add_....._section
- The more sections you add to your index the longer the search will take!

Nested tag searching is supported in interMedia Text.

Using Table CTX_OBJECTS and CTX_OBJECT_ATTRIBUTES View

The CTX_OBJECT_ATTRIBUTES view displays attributes that can be assigned to preferences of each object. It can be queried by all users.

Check out the structure of CTX_OBJECTS and CTX_OBJECT_ATTRIBUTE view, with the following DESCRIBE commands. Because we are only interested in

querying XML documents in this chapter, we focus on XML_SECTION_GROUP and AUTO_SECTION_GROUP.

```
Describe ctx_objects
  SELECT obj_class, obj_name FROM ctx_objects
  ORDRR BY obj_class, obj_name;
```

The result is:

```
...
SECTION_GROUP                  AUTO_SECTION_GROUP    <<==
SECTION_GROUP                  BASIC_SECTION_GROUP
SECTION_GROUP                  HTML_SECTION_GROUP
SECTION_GROUP                  NEWS_SECTION_GROUP
SECTION_GROUP                  NULL_SECTION_GROUP
SECTION_GROUP                  XML_SECTION_GROUP     <<==

...

Describe ctx_object_attributes
SELECT oat_attribute FROM ctx_object_attributes
  WHERE oat_object = 'XML_SECTION_GROUP';
```

The result is:

```
OAT_ATTRIBUTE
-------------
ATTR
FIELD
SPECIAL
ZONE

SELECT oat_attribute FROM ctx_object_attributes
  WHERE oat_object = 'AUTO_SECTION_GROUP';
```

The result is:

```
OAT_ATTRIBUTE
-------------
STOP
```

1 Create a Preference

The first thing you must do is create a preference. To do this, use the CTX_DDL.Create_Preference procedure.

For example:

CTX_DDL.Create_Preference

```
CTX_DDL.Create_Preference (
 preference_name  => 'books' /* or whatever you want to call it */
 object_name      => 'XML_SECTION GROUP' /* either XML_SECTION_GROUP or AUTO_
SECTION_GROUP */);
```

To drop this preference use the following syntax:

```
CTX_DDL.Drop_Preference (
 preference_name => 'books');
```

2 Set the Preference's Attributes

To set the preference's attributes for XML_SECTION_GROUP, use the following procedures:

- Add_Zone_Section
- Add_Attr_Section
- Add_Field_Section
- Add_Special_Section

To set the preference's attributes for AUTO_SECTION_GROUP, use the following procedures:

- Add_Zone_Section
- Add_Attr_Section
- Add_Field_Section

There are corresponding CTX_DDL.drop sections and CTX_DDL.remove section syntax.

2.1 Using CTX_DDL.add_zone_section

The syntax for CTX_DDL.add_zone_section follows:

```
CTX_DDL.Add_Zone_Section (
 group_name      => 'my_section_group' /* whatever you called it above */
 section_name    => 'author' /* what you want to call this section */
 tag             => 'my_tag' /* what represents it in XML */ );
```

where 'my_tag' implies opening with <my_tag> and closing with </my_tag>.

add_zone_section Guidelines

add_zone_section guidelines are listed here:

- Call CTX_DDL.Add_Zone_Section for each tag in your XML document that you need to search on.

2.2 Using CTX_DDL.Add_Attr_Section

The syntax for CTX_DDL.add_attr_section follows:

```
CTX_DDL.Add_Attr_Section ( /* call this as many times as you need to describe
                              the attribute sections */
 group_name      => 'my_section_group' /* whatever you did call it above */
 section_name    => 'author' /* what you want to call this section */
 tag             => 'my_tag' /* what represents it in XML */ );
```

where 'my_tag' implies opening with <my_tag> and closing with </my_tag>.

add_attr_section Guidelines

add_attr_section guidelines are listed here:

- Consider meta_data attribute author:

  <meta_data author = "John Smith" title="How to get to Mars">

The more sections you add to your index, the longer your search will take.

add_attr_section adds an attribute section to an XML section group. This procedure is useful for defining attributes in XML documents as sections. This allows you to search XML *attribute* text with the WITHIN operator.

The section_name:

- Is the name used for WITHIN queries on the attribute text.
- Cannot contain the colon (:) or dot (.) characters.
- Must be unique within group_name.
- Is case-insensitive.
- Can be no more than 64 bytes.

The tag specifies the name of the attribute in tag@attr format. This is case-sensitive.

Note: In the add_attr_section procedure, you can have many tags all represented by the same section name at query time. Explained in another way, the names used as the arguments of the keyword WITHIN can be different from the actual XML tag names. That is many tags can be mapped to the same name at query time. This feature enhances query usability.

2.3 Using CTX_DDL.add_field_section

The syntax for CTX_DDL.add_field_section follows:

```
CTX_DDL.Add_Field_Section (
 group_name     => 'my_section_group' /* whatever you called it above */
 section_name   => 'qq' /* what you want to call this section */
 tag            => 'my_tag' /* what represents it in XML */ );
 visible        => TRUE or FALSE );
```

add_field_section Guidelines

add_field_section guidelines are listed here:

- add_field_section and add_zone_section attributes differ in performance.
- In a document, tags can be repeated two or more times, however some tags, such as "title", occur only once. A DTD (or XML Schema) define how many times the tags occur.
- *Visible attribute*: This is available in add_field_section but not available in the add_zone_section. If VISIBLE is set to TRUE then a query such as "... CONTAINS cat... becomes irrelevant if cat occurs in title or paragraph.

  Consider again the query, "... CONTAINS cat...". You may not get a hit if you use VISIBLE=TRUE. If VISIBLE=FALSE, the index will be smaller. You may lose some functionality but your performance will be improved, compared to if you set VISIBLE =TRUE.

  - If you set up your index using the add_zone_section....
  - If you set up your index using the add_field_section....

Note: Constructing an index is harder if you have to cater for the fact that there could be more than one occurrence of any one tag.

- If the tag in your XML document occurs only once, use the single add_field_section procedure. For example, ".... CONTAINS cat and dog WITHIN title.....
- If the tag in your XML document occurs more than once, use the add_zone_section procedure. For example, ".... CONTAINS cat and dog WITHIN paragraph....". This has many possibilities.

How Attr_Section Differs From Field_Section

Attribute section differs from field section in the following ways:

- *Attribute text is considered invisible*, though, so the following:

  ```
  WHERE CONTAINS (..., '... jeeves',...)...
  ```

  does NOT find the document, somewhat like field sections. Unlike field sections, however, attribute section within searches can distinguish between occurrences. Consider the document:

  ```
  <comment author="jeeves">
    I really like interMedia Text
  </comment>
  <comment author="bertram">
    Me too
  </comment>
  ```

  the query:

  ```
  WHERE CONTAINS (...,'(cryil and bertram) WITHIN author', ...)...
  ```

  will NOT find the document, because "jeeves" and "bertram" do not occur within the SAME attribute text.

- *Attribute section names cannot overlap with zone or field section names*, although you can map more than one tag@attr to a single section name. Attribute sections do not support default values. Given the document:

  ```
  <!DOCTYPE foo [
    <!ELEMENT foo (bar)>
    <!ELEMENT bar (#PCDATA)>
  ```

```
<!ATTLIST bar
  rev CDATA "8i">
]>
<foo>
  <bar>whatever</bar>
</foo>
```

and attribute section:

```
ctx_ddl.add_attr_section('mysg','barrev','bar@rev');
```

the query:

8i within barrev does not hit the document, although in XML semantics, the "bar" element has a default value for its "rev" attribute.

2.4 Using CTX_DDL.add_special_section

The syntax for CTX_DDL.add_special_section follows:

```
CTX_DDL.Add_Special_Section (
 group_name     => 'my_section_group' /* whatever you called it above */
 section_name   => 'qq' /* what you want to call this section */ );
```

add_special_section Guidelines

add_special_section guidelines are listed here:

Here the tag option is not present. A section not defined with an open and close tag has an implicit definition.Use this section when your document is composed, for example, of mostly non-tagged sentences and paragraphs that you need to search. These are explicitly defined.

For example, if your query could be "... CONTAINS cat and dog WITHIN sentence"...

2.5 Using CtX_DDL.Add_Stop_Section

```
CtX_DDL.Add_Stop_Section (
 group_name     => 'my_section_group' /* whatever you called it above */
 section_name   => 'qq' /* what you want to call this section */ );
```

3 Creating Your Query Syntax

See the section, "Querying with the CONTAINS Operator" for information about how to use the CONTAINS operator in query statements.

Querying Within Attribute Sections

You can query within attribute sections when you index with either XML_SECTION_GROUP or AUTO_SECTION_GROUP as your section group type.

Assume you have an XML document as follows:

```
<book title="Tale of Two Cities">It was the best of times.</book>
```

You can define the section title@book as the attribute section title. You can do so with the CTX_DLL.Add_Attr_Section procedure or dynamically after indexing with ALTER INDEX.

Note: When you use the AUTO_SECTION_GROUP to index XML documents, the system automatically creates attribute sections and names them in the form attribute@tag.

If you use the XML_SECTION_GROUP, you can name attribute sections anything with CTX_DDL.Add_Attr_Section.

To search on Tale within the attribute section title, issue the following query:

```
WHERE CONTAINS (...,'Tale WITHIN title', ...)
```

Using XML_SECTION_GROUP and add_attr_section to Aid Querying

Consider an XML file that defines the BOOK tag with a TITLE attribute as follows:

```
<BOOK TITLE="Tale of Two Cities">
It was the best of times. </BOOK>
<Author="Charles Dickens">
Born in England in the town, Stratford_Upon_Avon </Author>
```

Recall the CTX_DDL.Add_Attr_Section syntax is:

```
CTX_DDL.Add_Attr_Section ( group_name, section_name, tag );
```

To define the title attribute as an attribute section, create an XML_SECTION_GROUP and define the attribute section as follows:

```
ctx_ddl_create_section_group('myxmlgroup', 'XML_SECTION_GROUP');
```

```
    ctx_ddl.add_attr_section('myxmlgroup', 'booktitle', 'book@title');
    ctx_ddl.add_attr_section('myxmlgroup', 'authors', 'author');
    end;
```

When you define the TITLE attribute section as such and index the document set, you can query the XML attribute text as follows:

```
... WHERE CONTAINS (...,'Cities WITHIN booktitle', ....)...
```

When you define the AUTHOR attribute section as such and index the document set, you can query the XML attribute text as follows:

```
... WHERE 'England WITHIN authors'
```

interMedia Text Example 4: Querying a... Document

This example does the following:

1. Creates and populates table res_xml
2. Creates an index, section_group, and preferences
3. Paramaterizes the preferences
4. Runs a test query against res_xml

```
drop table res_xml;

CREATE TABLE res_xml (
  pk            NUMBER PRIMARY KEY ,
  text          CLOB
  ) ;

insert into res_xml values(111,
  'ENTITY chap8 "Chapter 8, <q>Keeping it Tidy: the XML Rule Book </q>"> this is
the the document section');
commit;

---
--- script to create index on res_xml
---

--- cleanup, in case we have run this before
DROP INDEX res_index ;
EXEC CTX_DDL.DROP_SECTION_GROUP ( 'res_sections' ) ;

--- create a section group
```

```
BEGIN
  CTX_DDL.CREATE_SECTION_GROUP ( 'res_sections', 'XML_SECTION_GROUP' ) ;
  CTX_DDL.ADD_FIELD_SECTION ( 'res_sections', 'chap8', '<q>') ;
END ;
/

begin
  ctx_ddl.create_preference
    (
      preference_name => 'my_basic_lexer',
      object_name     => 'basic_lexer'
    );
  ctx_ddl.set_attribute
    (
      preference_name => 'my_basic_lexer',
      attribute_name  => 'index_text',
      attribute_value => 'true'
    );
  ctx_ddl.set_attribute
    (
      preference_name => 'my_basic_lexer',
      attribute_name  => 'index_themes',
      attribute_value => 'false');
end;
/

CREATE INDEX res_index
  ON res_xml(text)
  INDEXTYPE IS ctxsys.context
  PARAMETERS ( 'lexer my_basic_lexer SECTION GROUP res_sections' ) ;
```

Test the above index with a test query, such as:

```
SELECT pk FROM res_xml WHERE CONTAINS( text, 'keeping WITHIN chap8' )>0 ;
```

interMedia Example 5: Creating an Index and Performing a Text Query

Creating Table explain_ex to Use in this Example

```
drop table explain_ex;

create table explain_ex
  (
```

```
    id        number primary key,
    text      varchar(2000)
  );

insert into explain_ex ( id, text )
  values ( 1, 'thinks thinking thought go going goes gone went' || chr(10) ||
              'oracle orackle oricle dog cat bird'              || chr(10) ||
              'President Clinton' );
insert into explain_ex ( id, text )
  values ( 2, 'Last summer I went to New England'               || chr(10) ||
              'I hiked a lot.'                                  || chr(10) ||
              'I camped a bit.' );
commit;
```

Creating an Index for themes

```
begin
  Ctx_Ddl.Drop_Preference ('my_lexer');
end;
/
begin
  Ctx_Ddl.Create_Preference ( 'my_lexer', 'basic_lexer');
  Ctx_Ddl.Set_Attribute     ( 'my_lexer', 'index_text',   'true' );

  /* Experiment with 'index_themes' = 'false' */
  Ctx_Ddl.Set_Attribute     ( 'my_lexer', 'index_themes', 'true' );
end;
/

begin
  Ctx_Ddl.Drop_Stoplist ( 'my_stoplist' );
end;
/

begin
  Ctx_Ddl.Create_Stoplist ( 'my_stoplist' );
  Ctx_Ddl.Add_Stopword
    (
      stoplist_name => 'my_stoplist',
      stopword      => 'because'
    );
  Ctx_Ddl.Add_Stopword ( 'my_stoplist', 'and' );
  Ctx_Ddl.Add_Stopword ( 'my_stoplist', 'in' );
  Ctx_Ddl.Add_Stopword ( 'my_stoplist', 'to' );
  Ctx_Ddl.Add_Stopword ( 'my_stoplist', 'a' );
```

```
  Ctx_Ddl.Add_Stopword ( 'my_stoplist', 'I' );
end;
/

drop index explain_ex_text;
select err_text from ctx_user_index_errors;
create index explain_ex_text on explain_ex ( text )
  indextype is ctxsys.context
  parameters ( 'lexer my_lexer stoplist ctxsys.empty_stoplist' );
select err_text from ctx_user_index_errors;

begin
  Ctx_Ddl.Drop_Preference ('my_lexer');
  Ctx_Ddl.Drop_Stoplist ( 'my_stoplist' );
end;
/
```

Text Query Using "ABOUT" in the Text Query Expression

```
Set Define Off
select text
  from explain_ex
  WHERE CONTAINS ( text,
  '( $( think & go ) , ?oracle ) & ( dog , ( cat & bird ) ) & about(mammal
                                                    during Bill Clinton)' ) > 0;

select text
  from explain_ex
  WHERE CONTAINS ( text, 'about ( camping and hiking in new england )'  ) > 0;
```

Creating Sections in XML Documents that are Document Type Sensitive

Consider an XML document set that contains the <book> tag declared for different document types. You need to create a distinct book section for each document type. Assume that mydocname is declared as an XML document type (root element) as follows:

```
<!DOCTYPE mydocname ... [...
```

Within mydocname, the element <book> is declared. For this tag, you can create a section named mybooksec that is sensitive to the tag's document type as follows:

```
begin
ctx_ddl.create_section_group('myxmlgroup', 'XML_SECTION_GROUP');
ctx_ddl.add_zone_section('myxmlgroup', 'mybooksec', 'mydocname(book)');
end;
```

Note:

- Oracle8i knows what the end tags look like from the group_ type parameter you specify when you create the section group. The start tag you specify must be unique within a section group.
- Section names need not be unique across tags. You can assign the same section name to more than one tag, making details transparent to searches.

Repeated Sections

Zone sections can repeat. Each occurrence is treated as a separate section. For example, if <H1> denotes a heading section, they can repeat in the same documents as follows:

```
<H1> The Brown Fox </H1>
<H1> The Gray Wolf </H1>
```

Assuming that these zone sections are named Heading.

The query:

```
WHERE CONTAINS (..., 'Brown WITHIN Heading', ...)...
```

returns this document.

But the query:

```
WHERE CONTAINS (...,' (Brown and Gray) WITHIN Heading',...)...
```

does not.

Overlapping Sections

Zone sections can overlap each other. For example, if <B> and <I> denote two different zone sections, they can overlap in document as follows:

```
plain <B> bold <I> bold and italic </B> only italic </I>  plain
```

Nested Sections

Zone sections can nest, including themselves as follows:

```
<TD>
  <TABLE>
  <TD>nested cell</TD>
  </TABLE>
</TD>
```

Using the WITHIN operator, you can write queries to search for text in sections within sections.

Nested Section Query Example

For example, assume the BOOK1, BOOK2, and AUTHOR zone sections occur as follows in documents doc1 and doc2:

doc1:

```
<book1><author>Scott Tiger</author> This is a cool book to read.</book1>
```

doc2:

```
<book2> <author>Scott Tiger</author> This is a great book to read.</book2>
```

Consider the nested query:

```
'Scott WITHIN author WITHIN book1'
```

This query returns only doc1.

Presenting the Results of Your Query

A Text query application allows you to view the documents returned by a query. You typically select a document from the hitlist and then your application presents the document in some form.

With interMedia Text, you can render a document in different ways. For example, with the query terms highlighted. Highlighted query terms can be either the words of a word query or the themes of an ABOUT query in English. This rendering uses the CTX_DOC.HIGHLIGHT or CTX_DOC.MARKUP procedures.

You can also obtain theme information from documents with the CTX_DOC.THEMES PL/SQL package. Besides these there are several other CTX_DOC procedures for presenting your query results.

> **See Also:** *Oracle8i interMedia Text Reference* f for more information on the CTX_DOC PL/SQL package.

Frequently Asked Questions (FAQs): *inter*Media Text

Inserting XML data and Searching with interMedia Text

Question

Although InterMedia doesn't understand the hierarchical XML structure, can I do something like this...

```
<report>
<day>yesterday</day> there was a disaster <cause>hurricane</cause>
</report>
```

I would like to search the LOB's where cause was hurricane, is this possible?

Answer

You can perform that level of searching with the current release of interMedia. Currently to break a document up you would have to use our XML Parser with XSLT to create a stylesheet that transforms the XML into DDL. *i*FS gives you a higher level interface.

Another technique is to use a JDBC program to insert the text of the document or document fragment into a CLOB or LONG column, then do the searching using the CONTAINS() operator after setting up the indexes...

interMedia Text: Handling Attributes

Question

Currently *inter*Media Text has the option to create indexes based on the content of a section group. But most XML Elements are of the type of Element. So, the only option for searching would be attribute values. So, I am wondering if there is any way to build indexes on attribute values.

Answer

Releases from 8.1.6 and higher allow attribute indexing. See the following site: http://technet.oracle.com/products/intermedia/htdocs/text_training_816/Samples/imt_816_techover.html#SCN

CTXSYS/CTXSYS id and password

Question

We are installing the XSQL demos at http://technet.oracle.com//tech/xml/xsql_servlet/htdocs/relnotes.htm#ID3376.

At step 3, we are unable to access the database via user/password CTXSYS/CTXSYS. We cannot access this via running SQLPLUS from the web server's root directory or by accessing SQL PLUS normally. We can access either SCOTT/TIGER or SYSTEM/MANAGER via either method. Is there a step we missed somewhere where we need to add this user/permissions, etc.? If there is, can you tell us where to find the instructions on this? The error message we receive is:

```
ERROR:OCA-30017: error logging on to non-Oracle database[POL-5246] User does not exist.
```

We are using Oracle 8iLite, Win NT 4 SP 4.

Answer

Oracle8*i* Lite does not support Intermedia Text, so you can ignore this step. CTXSYS/CTXSYS is the default username/password for the Intermedia Text schema owner. The demos are designed to run against a regular Oracle8*i* database, so you may encounter other problems running them with Oracle8*i* Lite. The Servlet works fine for reading data out of Oracle8i Lite, however, it's just that the demos are not targeting the "lite" version.

Querying an XML Document

Question

I know that an intact XML documents are stored in a CLOB or BLOB with ORACLE XML solution.

1. XML documents stored in a CLOB/BLOB are able to be queried like table schema? For example:

   ```
   [XML document stored in BLOB]...<name id="1111"><first>lee</first>
   <sencond>jumee</second></name>...
   ```

   Is value(lee, jumee) able to be queried by elements, attributes and structure of XML document?

2. If some element or attribute is inserted / updated / deleted, all document must be updated? Or can insert / update / delete like table schema?
3. About locking, if someone manages an XML document stored in a BLOB / CLOB, nobody can access the same XML document? Is this true?

Answer

1. Using *inter*Media Text, you can find this document with a query like this:

   ```
   lee within first or this:jumee within second or this:1111 within name@id
   ```

   you can combine these like this:

   ```
   lee within first and jumee within secondor this:(lee within first) within
   name.
   ```

   For more information, please read the "*inter*Media Text Technical Overview" for 8.1.5 and 8.1.6 available on OTN.

2. *inter*Media Text indexes CLOB / BLOB, and this has no knowledge about XML specifically, so you cannot really change individual elements. You have to edit the document as a whole.
3. Just like any other CLOB, if someone is writing to the CLOB, they have it locked and nobody else can write to the CLOB. Other users can READ it, but not write. This is basic LOB behavior.

   Another alternative is to decompose the XML document and store the information in relational fields. Then you could modify individual elements, have element-level simultaneous access, and so on. In this case, using something called the USER_DATASTORE, you can use PL / SQL to reconstitute the document to XML for text indexing. Then, you get text search as if it were XML, but data management as if it were relational data. Again, see interMedia Text Technical Overview for more information.

*inter*Media Text and Oracle8i

Question

Is *inter*Media Text included in Oracle8*i*? What is the name of this package? Does the package insert and search XML documents into the database?

Answer

Context Cartridge is now called interMedia Text and is part of the Oracle8*i* *inter*Media option. Details are at http://www.oracle.com/database/options/intermedia.html. *inter*Media Text will not help you insert XML documents into the database, only search them.

*inter*Media XML Indexing

Question

Is it possible for *inter*MediaText to index XML such as:

2/7/1968

and then process a query such as:

Who has brown hair, that is, select name from person where hair.color = "BROWN"

Answer

Searches based on structural conditions are not yet available through *inter*Media Text. Attribute searches are supported from release 8.1.6. For reference you should not put data in attributes as that will not be compliant with XML Schema when it becomes a recommendation.

Searching CLOBs Using *inter*Media Text

Question

How would I define *inter*Media parameters so that I would be able to search my CLOB column for records that contained "aorta" and "damage". For example using the following XML (DTD implied):

```
WellKnownFileName.gif   echocardiogram   aorta
```

This is an image of the vessel damage. It would be nice to see a simple (or complicated) example of an XML *inter*Media implementation.

I assume there is no need to setup the ZONE or FIELDS.....Is this the case?

Answer

If you save an XML Document fragment in a CLOB, and enable an *inter*Media Text XML index on it, then you can do a SQL query which uses the CONTAINS() operator as the following query does:

Assume you have a document like an insurance claim...

```
77804
1999-01-01 00:00:00.0                8895                      1044
Paul               Astoria
123 Cherry Lane         SF         CA         94132
1999-01-05 00:00:00.0           7600            JCOX
It was becase of Faulty Brakes
```

If you store the content as a document fragment in a CLOB, then you can do a query like the following (assuming everything else you store in relational tables):

```
REM Select the SUM of the amounts of
REM  all settlement payments approved by "JCOX"
REM  for claims whose  relates to Brakes.
select sum(n.amount) as TotalApprovedAmount
  from insurance_claim_view v, TABLE(v.settlements) n
  where n.approver = 'JCOX'
  and contains(damageReport,'Brakes within Cause') >
```

Managing Different XML Documents With Different DTDs: Storing and Searching XML in CLOBs -- interMedia Text

Question

It was suggested that I store XML in CLOBs and use the DOM or SAX to reparse the XML later as needed. I agree that this was the best solution for my problem (which was how to manage many different XML documents using many different DTDs in a document management system) The big problem was searching this document repository to locate relevant information.

This is where *inter*Media Text seems ideal. It would be nice to see an example of setting this up using intermedia in Oracle8i, demonstrating how to define the XML_SECTION_GROUP and where to use a ZONE as opposed to a FIELD etc.

For example:

How would I define Intermedia parameters so that I would be able to search my CLOB column for records that had the "aorta" and "damage" in the using the

following XML (DTD implied) WellKnownFileName.gif echo cardiogram aorta This is an image of the vessel damage

Answer

You can't do XML structure-based searches with *inter*Media. You can search for text within a given element, but nothing more complicated than that. It also does not do attributes.You could load up each doc with the DOMParser and search that way, but that wouldn't scale very well.We are working on a project with a similar requirement. We are resorting to creating columns in the table for each bit of xml data we want to do serious searching on and loading it up from an initial XML parse. Of course that doesn't help if you need to do structured searches on arbitrary elements.

Question 2

Releases from 8.1.6 allow searching within attribute text. That's something like: dog within book@author. We are working on attribute value sensitive search, more like the following:

dog within book[@author = "Eric"]:

```
begin  ctx_ddl.create_section_group('mygrp','basic_section_group');
  ctx_ddl.add_field_section('mygrp','keyword','keyword');
  ctx_ddl.add_field_section('mygrp','caption','caption');
end;
create index myidx on mytab(mytxtcolumn)indextype is ctxsys.contextparameters
('section group mygrp');
select * from mytab where contains(mytxtcolumn, 'aorta within keyword')>0;
options:
```

- Use XML section group instead of basic section group if your tags have attributes or you need case-sensitive tag detection
- Use zone sections instead of field sections if your sections overlap, or if you need to distinguish between instances. For instance, keywords. If keywords is a field section, then (aorta and echo cardiogram) within keywords finds the document. If it is a zone section, then it does not, because they are not in the SAME instance of keywords.

It is not so clear. It looks to me like his example is trying to find instances of elements containing "damage" that have a sibling element containing "aorta" within the same record. It's not clear what exactly he means by "record".

If each record equates to the in his example, and there can be multiple records in a single XML LOB, than I don't see how you could do this search with *inter*Media.

If there is only one per CLOB/row, than perhaps you could find it by ANDing two context element queries. But that would still be a sloppy sort of xml search relying on some expected limitations of the situation more so than the structural composition actually called for.

Answer 2

What I meant by record was the obvious thing. The whole XML example was stored in a CLOB column in a table, therefore the Record was the row in the table that contained the XML code.

*inter*Media Text Role (ORA-01919: role 'CTXSYS' does not exist)

Question

With reference to your documentation, Oracle8i *inter*Media Text Migration, Part No. A67845-01, section "Roles and Users", it says Oracle8i *inter*Media Text provides the two roles for system administrators and application developers as CTXSYS Role and CTXAPP Role.

But when I issue a GRANT command to grant the CTXSYS role to a user it says ORA-01919: role 'CTXSYS' does not exist. When I queried the sys.dba_roles view, it does not give CTXSYS role.

Could you please reply me and help me out to solve the problem. I have installed the Oracle8I from the free CD shipped be you.

Answer

You might not have the *inter*Media Text installed? Did you take the "starter" database, or create one from scratch? If the latter, did you select *inter*Media Text during the install?

Searching XML Documents and Returning a Zone

Question

I need to store a large XML file in Oracle8*i*, search it, and return a specific tagged area. I have not found a clear way to store a large XML file, index it, allow searching

on it AND return Tagged sections from it based on a search.Using *inter*Media Text some of this is possible:

- I can store an XML file in a CLOB field
- I can index it with ctxsys.context
- I can create <Zones> and <Fields> to represent the Tags in my XML fileEx. ctx_ddl.add_zone_section(xmlgroup,"dublincore", dc);
- I can search for text within a Zone or fieldEx. Select title from mytable where CONTAINS(textField,"some words WITHIN dublincore")

What I need to know is how do I return a zone or a field based on a text search?

Answer

*inter*Media Text will only return the "hits". You will need to subsequently parse the CLOB to extract a section.

Storing an XML Document in CLOB: Using *inter*Media Text

Question

I need to store XML files(that are present on the file system as of now) into the database. I want to store the whole document. What I mean is that I do not want to break the document as per the tags and then store the info in separate tables / fields. Rather I want that I should have a universal table, that I can use to store different XML documents. I think internally it will be stored in a CLOB type of field in my case. My XML files will always contain ASCII data.

Can this be done using *inter*Media. Should we be using *inter*Media Text or *inter*Media Annotator for this? I downloaded Annotator from OTN, but I could not store XML document in the database.

I am trying to store XML document into CLOB column. Basically I have one table with the following definition(shown in red color below):

```
CREATE TABLE xml_store_testing
(
    xml_doc_id  NUMBER,
    xml_doc     CLOB  )
```

I want to store my XML document in xml_doc field.

I have written another PL/SQL procedure shown below, to read the contents of the XML Document. The XML document is available on the file system. XML document contains just ASCII data - no binary data.

```
CREATE OR REPLACE PROCEDURE FileExec
(
  p_Directory      IN VARCHAR2,
  p_FileName       IN VARCHAR2)
  AS   v_CLOBLocator  CLOB;
       v_FileLocator       BFILE;
BEGIN
    SELECT  xml_doc
    INTO      v_CLOBLocator
    FROM    xml_store_testing
    WHERE  xml_doc_id = 1
    FOR        UPDATE;
    v_FileLocator := BFILENAME(p_Directory, p_FileName);
    DBMS_LOB.FILEOPEN(v_FileLocator, DBMS_LOB.FILE_READONLY);
    dbms_output.put_line(to_char(DBMS_LOB.GETLENGTH(v_FileLocator)));
      DBMS_LOB.LOADFROMFILE(v_CLOBLocator, v_FileLocator,
      DBMS_LOB.GETLENGTH(v_FileLocator));
      DBMS_LOB.FILECLOSE(v_FileLocator);
END FileExec;
```

Answer

Put the XML documents into your CLOB column, then add an *inter*Media Text index on it using the XML section-group. See the documentation and overview material at http://technet.oracle.com/products/intermedia.

Question 2

When I execute this procedure, it executes successfully. But when I select from the table I see unknown characters in the table in CLOB field. Could this be because of the reason of the character set difference between operating system (where XML file resides) and database (where CLOB data resides).

Answer 2

Yes. If the character sets are different then you probably have to pass the data through UTL_RAW.CONVERT to do a character set conversion before writing to the CLOB.

Loading XML Documents into the Database and Searching with *inter*Media Text

Question

How do I insert XML documents into a database?

Specifically I need to insert the XML document "as is" in column of datatype CLOB into a table.

Answer

Oracle's XML-SQL Utility for Java offers a command-line utility that can be used for Loading XML data. More information can be found on the XML-SQL Utility at: http://technet.oracle.com/tech/xml

You can insert the XML documents as you would any text file. There is nothing special about an XML-formatted file from a CLOB perspective.

Question 2

I understand that Oracle *inter*Media Text can be used to index and search XML stored in CLOBs. Is this true? Any advice on how to get started with this?

Answer 2

Prior versions of *inter*Media Text only allowed tag-based searching. The current version, Release 3 (8.1.7) of Oracle8i, allows for XML structure and attribute based searching. There is documentation on how to have the index built and the SQL usage in the Oracle8i *inter*Media documentation.

See Also: *Oracle8i interMedia Text Reference.*

Searching XML with WITHIN Operator

Question

I have this xml:

```
<person>
  <name>efrat</name>
 <childrens>
     <child>
         <id>1</id>
         <name>keren</name>
     </child>
  </childrens>
```

```
</person>
```

How do I find the person who has a child name keren but not the person's name keren? Assuming I defined every tag with the add_zone_section that can be nested and can include themselves.

Answer

Use selectSingleNode or selectNodes with XPATH string as a parameter.eg. selectSingleNode("//child/name[.='keren'])Also, I recommend making id as an attribute instead of a tag.

*inter*Media Text and XML

Question

Where can I get good samples of using XML with *inter*Media.

Answer

See the following sites for more information:

http://technet.oracle.com/sample_code/products/intermedia/htdocs/text_samples/imt_815_techover.html and

http://technet.oracle.com/sample_code/products/intermedia/htdocs/text_samples/imt_816_techover.html

For the moment these are the best resources.There's also some new interMedia utilities and add-ons that could help you at:
http://technet.oracle.com/software/products/intermedia/software_index.htm

More XML samples ave been added to the 8.1.6 interMedia Text Doc, *Oracle8i interMedia Text Reference.*

*inter*Media Text and XML: Add_field_section

Question

Regarding XML with *inter*Media Text: Is there a way to feed an XML document into *inter*Media Text and have it recognize the tags, or do I have to use the add_field_section command for each tag in the XML document. My XML documents have hundreds of tags. Is there an easy way to do this?

Answer

Which version of the database are you using? I believe you need to do it for 8.1.5 but not 8.1.6.

You can use AUTO_SECTION_GROUP in 8.1.6

*inter*Media and XML Support

Question

Is there someone out there who can provide some real world examples of performing this simple task. I have an XML document that I want to feed into Oracle8i and search by content using tags. My XML document has over 100 tags, do I have to sit their and do an ADD_FIELD_SECTION for every tag....If not, where is this documented.

Answer

XSQL Servlet ships with a complete (albeit simple from the *inter*Media standpoint) example of a SQL script that creates a complex XML Datagram out of Object Types, and then creates an interMedia Text index on the XML Document Fragment stored in the "Insurance Claim" type.

If you download the XSQL Servlet, and look at the file ./xsql/demo/insclaim.sql you'll be able to see the interMedia stuff at the bottom of the file. One of the key new features in interMedia in 8.1.6 as outlined in one of the URL I posted in my previous reply is the AUTO Sectioner for XML. In 8.1.5, you do have to manually created your field sections.

Question 2

Is there a "Hello World" sample available anywhere? I am getting a javascript error on the XSQL servlet download page.

Answer 2

What follows is the content of the aforementioned demo file. It sets up the tables, types, and object views for the XSQL Insurance Claim Demo that you try live on our OTN demo site at http://technet.oracle.com/tech/xml/demo/demo1.htm

The *inter*Media Text-related part starts at the line that reads:

```
ctx_ddl.drop_preference()
```

In this example, an insurance claim has a "DamageReport" which is an XML Document fragment. The *inter*Media code at the end shows how to setup an XML searching index on the <CAUSE> and <MOTIVE> tags in this "DamageReport" document fragment.

```
set scan offset echo onset termout onREMREM $Author: smuench $REM $Date:
1999/11/27 14:48:10 $REM $Source: C:\\cvsroot/xsql/src/demo/insclaim.sql,v $REM
$Revision: 1.3 $REMdrop synonym claim;drop table settlement_payments;drop view
insurance_claim_view;drop table insurance_claim;drop view policy_view;drop table
policy;drop view policyholder_view;
drop table policyholder;drop type insurance_claim_t;
drop type settlements_t;
drop type payment;
drop type policy_t;
drop type policyholder_t;
drop type address_t;
create type address_t as object( Street varchar2(80), City Varchar2(80), State
VARCHAR2(80),Zip NUMBER );
./create type policyholder_t as object( CustomerId number,
   FirstName varchar2(80),
   LastName varchar2(80),
   HomeAddress address_t);
./create type policy_t as object(
  policyID number,  primaryinsured policyholder_t);
./create type payment as object(
  PayDate DATE,  Amount NUMBER,  Approver VARCHAR2(8));
./create type settlements_t as table of payment;
./create type insurance_claim_t as object (
  claimid number,filed   date,  claimpolicy policy_t,
  settlements settlements_t,  damageReport varchar2(4000) /* XML */);
./create table policyholder( CustomerId number,
  FirstName varchar2(80),   LastName varchar2(80),
  HomeAddress address_t,  constraint policyholder_pk primary key (customerid));
insert into policyholder values ( 1044, 'Paul','Astoria',
  ADDRESS_T('123 Cherry Lane','SF','CA','94132'));
insert into policyholder values ( 1045, 'Martina','Boyle',
  ADDRESS_T('55 Belden Place','SF','CA','94102'));
create or replace force view policyholder_view of policyholder_t
       with object OID
...
...
create or replace force view insurance_claim_view of insurance_claim_t
with object OID (claimid)
    as select c.claimid,c.filed,
       (SELECT value(pv)
```

```
        from policy_view pv
        WHERE pv.policyid = c.claimpolicy),
        CAST(MULTISET(SELECT PAYMENT(sp.paydate,sp.amount,sp.approver)
            as Payment from settlement_payments sp
            WHERE sp.claimid = c.claimid) AS settlements_t),c.damagereport
        from insurance_claim c;commit;
begin  ctx_ddl.drop_preference('Demo');
end;
/begin  ctx_ddl.create_preference('Demo', 'basic_lexer');
ctx_ddl.set_attribute ('Demo', 'index_themes', '0');
ctx_ddl.set_attribute ('Demo', 'index_text', '1');
ctx_ddl.create_section_group('demo_xml', 'xml_section_group');
ctx_ddl.add_zone_section('demo_xml', 'CAUSE', 'CAUSE');
ctx_ddl.add_zone_section('demo_xml', 'MOTIVE', 'MOTIVE');
end;
/create index
ctx_xml_i on insurance_claim(damagereport)indextype is
ctxsys.contextparameters('LEXER Demo SECTION GROUP demo_xml');
create synonym claim for insurance_claim_view;
```

Oracle8i Lite 4.0.0.2.0: *inter*Media Text is Not Supported

Question

I cannot initialize the database and run the SQL scripts for the demo programs:

1. I cannot connect as CTXSYS / CTXSYS. What is the source of the *inter*Media Text packages? Should user CTXSYS exist in a default installation?
2. I get a syntax error when attempting to execute GRANT QUERY REWRITE TO SCOTT.
3. I also get syntax errors when running some of the scripts, such as, airport.sql, although others, such as, index.sql, complete fine. I'm running a fresh installation of Oracle*8i* Lite 4.0.0.2.0.

Answer

1. *inter*Media Text is a feature of Oracle*8i*. If you're using Oracle*8i* Lite, then it is not available.
2. QUERY REWRITE is a privilege supported in Oracle8i, not Oracle8i Lite. So this failure is also explainable.

3. AIRPORT.SQL fails on the very last statement which is creating a FUNCTIONAL INDEX on UPPER(description). Functional indexes are an Oracle8i feature not available in 8i Lite. Also, if doing a fresh install, you will get some error messages during SQL script execution. If you look at the script, you'll find that it tries to delete a table before creating it. On a fresh install, these tables will not exist, so you'll get an error.

SQL in *inter*Media context

Question

I have an XML document that I have stored in CLOB. I have also created the indexes on the tags using section_group, and so on. One of the tags is <SALARY> </SALARY> I want to write an SQL statement so as to select all the records that have salary lets say > 5000.

How do I do this? I cannot use WITHIN operator. I want to interpret the value present in this tag as a number. This could be floating point number also since this is salary.

Answer

You can't do this in *inter*Media Text. Range search is not really a text operation. The best solution is to use the other Oracle XML parsing utilities to extract the salary into a NUMBER field -- then you can use *inter*Media Text for text searching, and normal SQL operators for the more structured fields, and achieve the same results.

XML and *inter*Media Text

Question

We are storing all our documents in XML format in a CLOB. Are there utilities available in Oracle perhaps *inter*Media to retrieve the contents a field at a time, that is given a field name, get the text between tags, as opposed to retrieving the whole document and traversing the structure?

Answer

*inter*Media does not do section extraction. See XM-SQL Utility for this.

Creating an Index on Three Columns?

Question

I have created a view based on 7-8 tables and it has columns like, custordnumber, product_dscr, qty, prdid,shipdate, ship_status, and so on. I need to create an *inter*Media index on the three columns:

- custordnumber
- product_dsc
- ship_status

Is there a way to create a text index on these columns?

Answer

The short answer is yes. You have two options:

1. Use the USER_DATASTORE object to create a concatenated field on the fly during indexing;
2. Concatenate your fields and store them in an extra CLOB field in one of your tables. Then create the index on the CLOB field. If you're using Oracle 8.1.6, then you also have the option of placing XML tags around each field prior to concatenation. This gives you the capability of searching WITHIN each field.

Searching Structured and Unstructured Data

Question

We need to insert data in the Database from an XML file. Currently we only can insert structured data with the table already created. Is this true?

We are working in a law project where we need to store laws that have structured data and unstructured data, and then search the data using *inter*Media text.

Can we insert unstructured data too? Or do we need to develop a custom application to do it? Then if we have the data stored with some structured parts and some unstructured parts, can we use *inter*Media Text to search it?

If we stored the unstructured part in a CLOB, and the CLOB has some tags, how can we search only the data in an specific tag?

Answer

Consider using *i*FS which allows you to break up a document storing it across tables and in a LOB. Currently *inter*Media Text can perform data searches with tags but is not knowledgeable about the hierachical XML structure. From release 8.1.6, *inter*Media Text has this capability along with name/value pair attribute searches.

Question 2

So, if I understand your answer this document breaking is not possible in these moments if I don't create a custom development? Although *inter*Media does not understand hierachical XML structure, can I do something like this?

```
<report>
   <day>yesterday</day> there was a disaster <cause>hurricane</cause>
</report>
```

Indexing with *inter*Media I would like to search the LOBs where cause was hurricane, is this possible?

Answer 2

You can perform that level of searching with the current release of *inter*Media Text. Currently to break a document up you would have to use the XML Parser with XSL-T to create a stylesheet that transforms the XML into DDL. *i*FS gives you a higher level interface.

Another technique is to use a JDBC program to insert the text of the document or document fragment into a CLOB or LONG column, then do the searching using the CONTAINS() operator after setting up the indexes.

6

Customizing Content with XML: Dynamic News Application

This chapter contains the following sections:

Introduction to the Dynamic News Application

It uses Oracle XML platform components together with the Oracle8i database to build a web-based news service.

The combination of Java, XML, XSL, HTML, and Oracle8i makes Dynamic News flexible and robust.

- With news items in the database, you can personalize content by executing queries based on user input.
- XML, XSL, and HTML allow you to customize the presentation for multiple platforms.
- The Dynamic News application pregenerates XML documents when possible to improve performance.

Problem: To customize news received at a browser according to user user requests.

Solution: The solution uses Oracle XML Components, Oracle8i database, and custom servlets. The solution is described in this chapter.

Oracle XML Components Used: XML Parser for Java, XML-SQL Utility (XSU) for Java

---

Dynamic News Main Tasks

Dynamic News application shows you how to do the following tasks:

- Store news headlines in the database
- Output the news in XML
- Apply XSL stylesheets to format new headlines

Overview of the Dynamic News Application

Dynamic News pulls news items (headlines) from the database to build HTML pages. The HTML pages are customized according to user preferences.

The pages present lists of items, with each item hyperlinked to a complete article. Each news item has attributes including:

- Category, such as Sports or Technology
- Subcategory, such as Baseball or Software
- Type, such as Feature or Review.

Three Levels of Customization: Static, Semi-Dynamic, and Dynamic

Dynamic News uses these attributes to offer three levels of customization:

- Static
- Semi-dynamic
- Dynamic

Table 6–1 describes these usage choices.

Table 6–1 Dynamic News: Three Levels of Customization

Customization Level	Description
Static	Static pages are not customized. An end-user at this level gets a page listing all items from each category, sub-category, and type. The news system administrator uses the Administration servlet to generate static XML documents periodically (for example, every hour on the hour). The application could build such pages on demand, but it's faster to serve up a pregenerated page than to run a query and build the same page for each user who requests it.
Semi-Dynamic	Semi-dynamic pages combine pregenerated lists of items. An end-user chooses one or more categories, and Dynamic News builds a page listing the items from those categories. The news admin uses the Administration servlet periodically to pregenerate the lists of items in each category. Like static pages, semi-dynamic pages are built from pregenerated documents to improve performance.
Dynamic	Dynamic pages are built when end-users request them. Content comes directly from the database; nothing is pregenerated. First, an end-user invokes a servlet to choose categories, sub-categories, and types. Next, Dynamic News queries the database for items matching that criteria and uses the result set to build an XML document. Then, as with static and semi-dynamic pages, it applies an XSLT transformation to generate HTML.

Note: The term "dynamic" and "static" refer to the page contents not its behavior.

Dynamic News SQL Example 1: Item Schema, nisetup.sql

Here's the SQL from nisetup.sql, that defines the structure of a news item:

```
CREATE TABLE news.NEWS_ITEMS
( ID   NUMBER NOT NULL,
  TITLE           VARCHAR2(200),
  URL             VARCHAR2(200),
  DESCRIPTION     VARCHAR2(2000),
  ENTRY_DATE      DATE,
  CATEGORY_ID     NUMBER,
  SUB_CATEGORY_ID NUMBER,
  TYPE_ID         NUMBER,
  SUBMITTED_BY_ID NUMBER,
  EXPIRATION_DATE DATE,  A
  APPROVED_FLAG   VARCHAR2(1)
);
```

Dynamic News Servlets

Table 6–2 lists the servlets used in the Dynamic News application. These servlets provide entry points to the application logic:

Table 6–2 Dynamic News Servlets

Servlet	Description	File Name
Administration	■ Adds news items to the database. ■ Maintains lists of users, types, and categories. ■ Generates XML and HTML for a static (non-customized) news page.	xmlnews/admin/AdminServlet.java
Semi-Dynamic	Generates lists of news items in categories chosen by the end-user.	xmlnews/dynamic/SemiDynamicServlet.java
Dynamic	Retrieves news items from the database to generate custom pages based on end-user preferences.	xmlnews/dynamic/DynamicServlet.java

How Dynamic News Works: Bird's Eye View

Generating XML Documents to Build HTML Pages

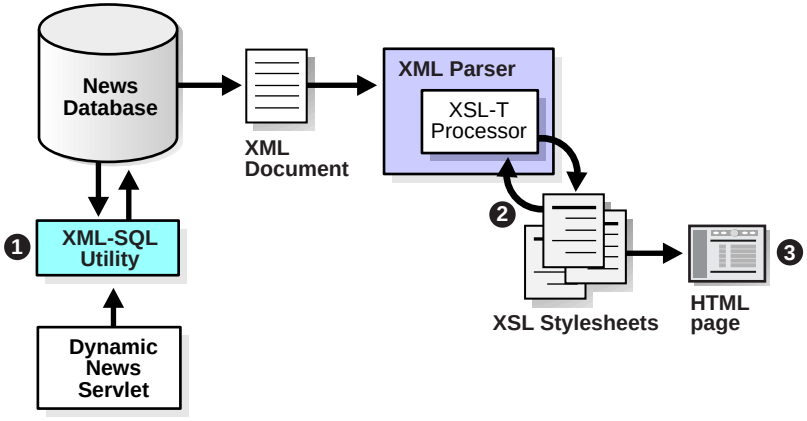
Dynamic News generates XML documents to build HTML pages:

- Static Pages: Built from XML documents pregenerated at intervals set by the news system administrator.
- Semi-Dynamic Pages: Built from pregenerated XML documents that list the items in categories chosen by the user.
- Dynamic Pages: Built on demand from XML documents that list items by categories, subcategories, and types chosen by the user.

Figure 6–1 gives an overview of how Dynamic News performs these steps:

1. Calls Oracle XML-SQL Utility (XSU) . This queries the database for news items and writes the results to an XML document. This happens as follows:
   - In batch mode for Static pages
   - In batch mode for Semi-Dynamic pages
   - On demand for Dynamic pages
2. Uses the XSL-T Processor of the Oracle XML Parser for Java to transform the XML into HTML via one of three XSL stylesheets: one for Netscape Navigator, one for Internet Explorer, or a general stylesheet for all other browsers.
3. Delivers the HTML page to the user through a Web server.

Figure 6–1 Dynamic News



Static Pages

Dynamic News generates static pages to display all available news items. These pages are built at intervals set by the news system administrator, for example, every hour on the hour; otherwise, they don't change.

When to Use Static Pages?

Static pages are useful in any application where data doesn't change very often. For example, when publishing daily summaries from ERP or customer applications. Because the content is static, it's more efficient to pregenerate a page than to build one for each user who requests it.

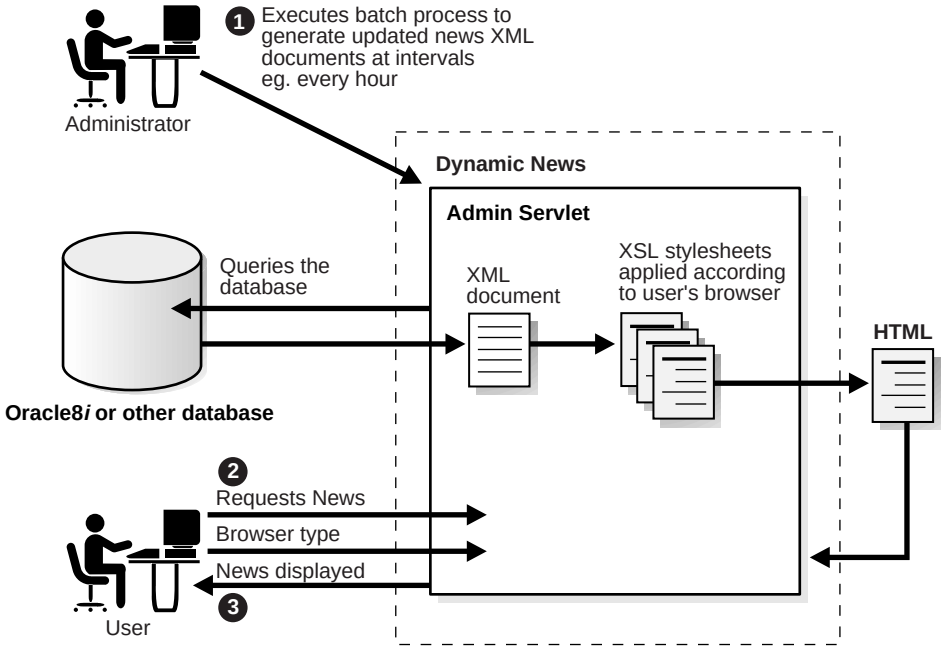
How Static Pages Works

The admin executes a batch process, implemented from the Administration servlet, that queries the database and generates an XML document. When an end-user invokes Dynamic News to display all news, a servlet gets the browser type from the user-agent header of the HTTP request, then reads the XML document, and applies the appropriate XSL stylesheet.

Finally, it returns an HTML page formatted for the end-user's browser, as shown in Figure 6–2.

Another approach would be to apply XSL stylesheets as part of the batch process, generating one HTML file for each stylesheet. In this case, you end up with more files to manage, but the runtime servlet is smaller.

Figure 6–2 Dynamic News: Static Pages - Generating XML Documents



Semi-Dynamic Pages

The application builds semi-dynamic pages by combining pregenerated lists. The lists of items per category are pregenerated by the administrator (one XML file for each category), but pages that contain them are customized for each user. End-users choose categories such as Sports, Business, and Entertainment.

When to Use Semi-Dynamic Pages

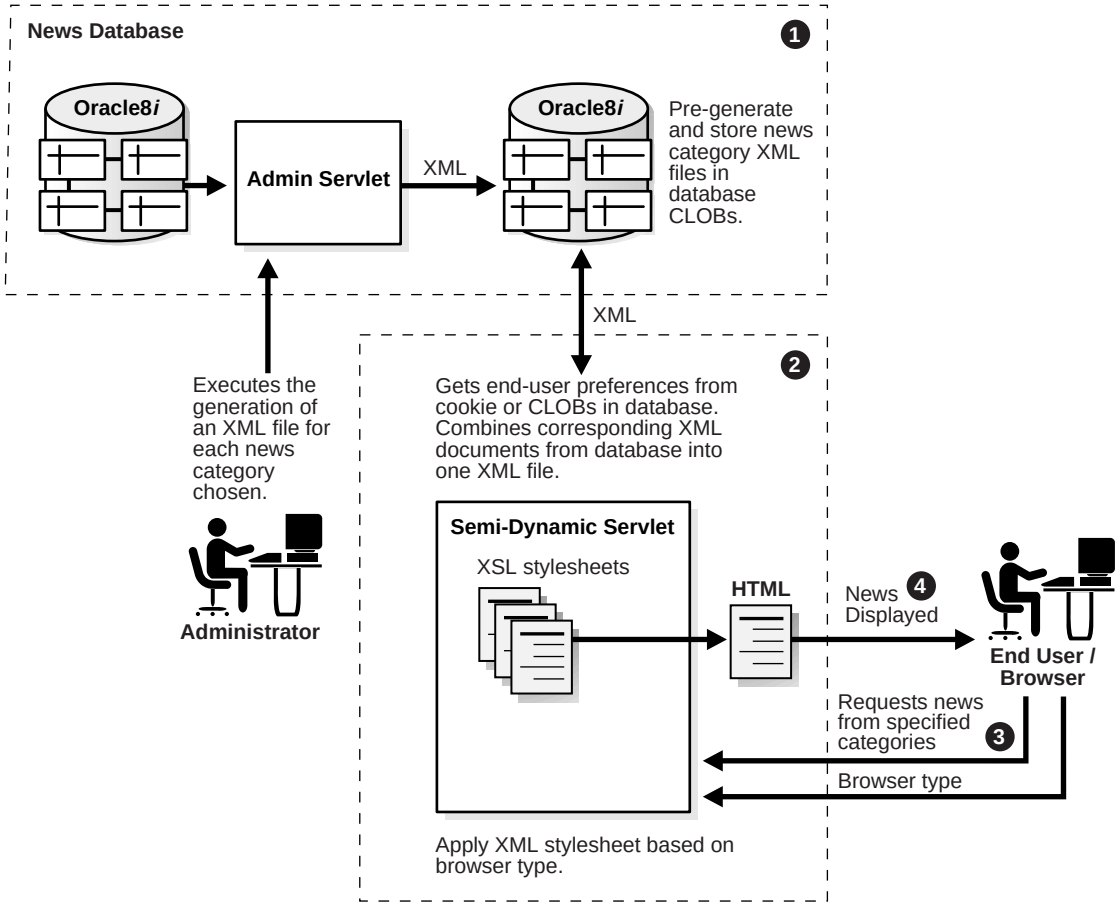
The semi-dynamic approach is useful when the data doesn't change very often and you want to give the end-user a relatively small number of choices. An application that offers more choices has to pregenerate more documents, and benefits degrade proportionally.

How Semi-Dynamic Pages Work

Figure 6–3 shows how semi-dynamic generation works. There are two phases:

- *Phase 1 - Static Processing Phase*: An administrator uses the Administration Servlet periodically to pregenerate XML files and store them in CLOBs in the database. You could also store them in a simple flat-file system, trading the benefits of the database for potential performance gains.
- *Phase 2 - Dynamic Processing Phase*: This phase begins when an end-user requests news items from specified categories. A servlet pulls CLOBs from the database and combines them into one XML document. It stores user preferences both in the database and in a client-side cookie, and reads them from the cookie where possible to improve performance. It then transforms the XML document into an HTML page using a XSL stylesheet matched to the end-user's browser. As with static pages, the servlet gets the browser type from the user-agent header of the HTTP request.

Figure 6–3 Dynamic News: Semi-Dynamic Pages - Generating XML Documents



Dynamic Pages

The application builds dynamic pages on demand by pulling items directly from the database. End-users access the "Create / Edit User Preference Page" to choose categories, subcategories, and types (for example, Entertainment - Movies - Review).

When to Use Dynamic Pages

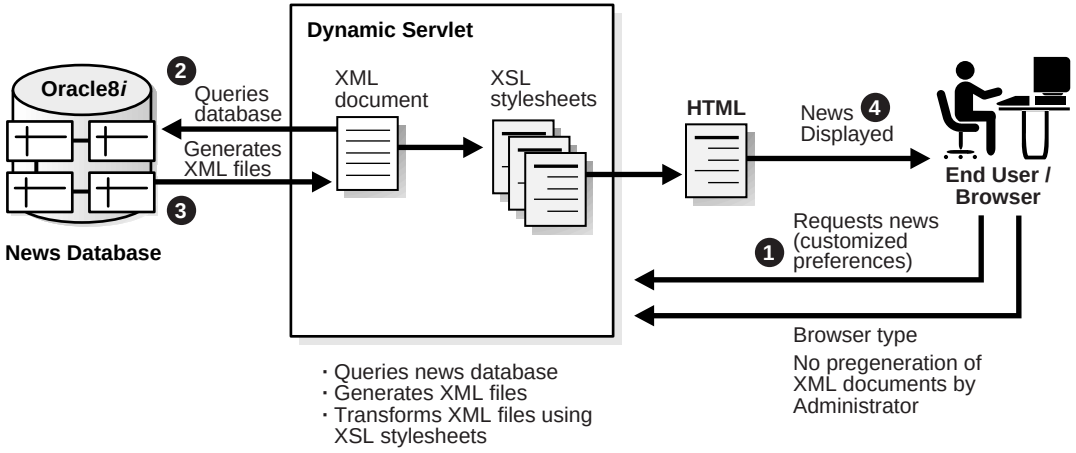
Dynamic pages are useful for delivering up-to-the-minute information, such as breaking news. They are also useful for delivering historical data, such as the closing price of any specified stock on any day in the last 10 years. It would be impractical to pregenerate documents for every possible request, but straightforward and efficient to pull the figures from the database.

How Dynamic Pages Works

Figure 6–4 shows how dynamic generation works. Unlike the other runtime models, the administrator does not pregenerate XML documents. Instead, the Dynamic Servlet queries the database for news items based on the end-user's customization choices.

The servlet stores user preferences both in the database and in a client-side cookie, and reads them from the cookie where possible to improve performance. Using the query results, the servlet generates an XML file and transforms it using an XSL stylesheet into an HTML page for the user's browser. As with the other approaches, the application gets the browser type from the user-agent header of the HTTP request.

Figure 6–4 Dynamic News: Dynamic Pages - Generatng XML Documents



Personalizing Content

Oracle8*i* makes Dynamic News flexible. Because news items are stored in the database, Dynamic News can customize content on demand. The code examples in this section show how the application personalizes pages by retrieving news items in categories specified by the end-user. The main tasks are:

1. Get end-user preferences.
2. Pull news items from the database.
3. Combine news items to build a document.
4. After assembling personalized content, the application customizes presentation of the page, formatting it for the end-user's browser as described later in this document.

1 Get End-User Preferences

Logic for processing preferences is distributed throughout the application, which stores the data both in the database and in client-side cookies. The application reads preference data from a cookie whenever possible to improve performance. If it can't get the data from a cookie (for example, because the end-user is visiting the site for the first time, or the end-user's browser does not accept cookies), the application reads preference data from the database.

From a Client-Side Cookie

The two methods below show how the application processes preference data stored in a cookie. Both methods come from xmlnews.common.UserPreference. Here's a sample cookie:

```
DynamicServlet=3$0$0#4$2$1***242
```

The cookie uses dollar signs to separate preference values, pound signs to separate categories, and three asterisks as a token to separate user ID and preference data. The sample cookie above shows that user 242 wants items from categories 3 and 4. In category 3, the user wants items of all types in all subcategories (a value of 0 selects all items). In category 4, the user wants items from subcategory 2 only, and within that subcategory, only items of type 1.

The sample app processes such cookies in two steps:

1. First, getNewsCookie gets the "DynamicServlet" cookie from the browser that issued the HTTP request.
2. Then loadPreferenceFromCookie parses it to get a String that contains that user's ID and preferences.

```
public Cookie getNewsCookie(HttpServletRequest request)
               throws Exception {
      Cookie c[] = request.getCookies();
      Cookie l_returnCookie = null;
               for (int i = 0; (c!= null) && (i < c.length); i++) {
               if (c[i].getName().equals("DynamicServlet")) {
                   l_returnCookie = c[i];
                   }
                 }
                 return l_returnCookie;
               }
public Vector loadPreferenceFromCookie(Cookie p_cookie) throws Exception {
      Vector l_prefId = new Vector(2);
```

```
            String l_Preferences = p_cookie.getValue();
            StringTokenizer l_stToken = new StringTokenizer(l_Preferences, "***");
            String l_userId = "";
            while (l_stToken.hasMoreTokens()) {
               // First Token is User Preference.
               l_Preferences = l_stToken.nextToken();
               // Second Token is User ID.
               l_userId = l_stToken.nextToken();
            }
            l_prefId.addElement(l_Preferences);
            l_prefId.addElement(l_userId);
            return l_prefId;
         }
```

Querying the Database

If it can't read preferences from a cookie, the application queries the database. The class `xmlnews.common.GenUtility` implements methods that connect to the database and fetch news categories, sub-categories, and types.

The semi-dynamic servlet and the dynamic servlet both call these methods and the methods `loadInitalPreference` and `constructUserPreference` . These are both implemented in xmlnews/common/UserPreference.java.

Method `loadInitalPreference` calls `getSubCategories`, then loops through the result set, combining category values with separator characters to build a preference string.

```
public String loadInitialPreference(Vector p_category, Vector p_subcategory,
        Vector p_types, Connection p_con)
throws Exception {
GenUtility m_general = new GenUtility();
...
for (int i = 0; i < p_category.size(); i++) {
        String l_cat[] = (String []) p_category.elementAt(i);
        l_category = l_cat[0];
        Vector l_subcategory = m_general.getSubCategories(p_con,l_cat[0]);

        for(int l_j = 0, l_k = 0; l_j < l_subcategory.size(); l_j++, l_k++)
 {
...
// Append the next preferences to the constructed string
   l_userPref = l_userPref+"#"+l_category+"$"+l_subCat+"$"+l_typeStr;
          }
   }
```

```
...
    return l_userPref;
   }

public static Vector getSubCategories(Connection p_conn, String p_categoryId)
    throws Exception {
    Vector l_subCats = new Vector();

PreparedStatement l_pstmt = p_conn.prepareStatement(
   "Select id, name from sub_categories where category_id = ? ");
    l_pstmt.setString(1, p_categoryId);
    ResultSet l_rset = l_pstmt.executeQuery();

while (l_rset.next()) {
     String[] l_subCat = new String[2];
     l_subCat[0] = new String(l_rset.getString(1));
     l_subCat[1] = new String(l_rset.getString(2));
     l_subCats.addElement(l_subCat);
     }
l_pstmt.close();
return l_subCats;
}
```

For example, the following code comes from `xmlnews.dynamic.DynamicServlet.service`.

It calls these methods to read end-user preferences from the database, then uses the preferences to build an HTML page.

```
public void service(HttpServletRequest p_request,
      HttpServletResponse p_  response)
      throws ServletException {

   // The following are declared elsewhere as class variables
   // and initialized in the servlet's init method.
   // GenUtility m_general = null;

   // m_general = new GenUtility();
   // UserPreference m_userPreference = null;
   // m_userPreference = new UserPreference();
   ...

   // If the database connection has been closed, reopen it.
      if (m_connection == null || m_connection.isClosed())
          m_connection = m_general.dbConnection();
   ...
```

```
      String l_preference = m_userPreference.loadInitialPreference(
      m_general.getCategories(m_connection),
      null, m_general.getTypes(m_connection),
                              m_connection);

m_userPreference = m_userPreference.constructUserPreference
   ( l_preference,m_status);

   // Display the Dynamic Page
      this.sendDynamicPage(l_browserType, p_response,
      l_userName, m_userPreference,
      m_servletPath + "?REQUEST_TYPE=SET_ADVANCED_USER_PREFS",
      m_servletPath + "?REQUEST_TYPE=LOGIN_REQUEST",
      m_servletPath + "?REQUEST_TYPE=LOG_OUT_REQUEST",
      m_servletPath);
    ...
 }
```

2 Pull News Items from the Database

The following code, from `xmlnews.admin.AdminServlet.performGeneration` and `xmlnews.admin.AdminServlet.staticProcessingHtml`, shows how the application queries the database for news items in each available category and converts each result set to a XML document.

The database stores the XML for each category as a CLOB (Character Large OBject), so the application can handle very long lists.

```
public void performGeneration(String p_user, String p_genType,
      HttpServletResponse p_response)
      throws ServletException, IOException {
...
      try {
      String l_fileSep = System.getProperty("file.separator");
      String l_message = ""; // Holds status message

      if (p_genType.equals("BATCH_GEN")) { // Batch Generation
      String l_htmlFile = "BatchGeneration";
      String l_xslFile  = "BatchGeneration";
      String l_xmlFile  = "BatchGeneration";

      // Generate the XML and HTML content and save it in a file
         this.staticProcessingHtml(
         m_dynNewsEnv.m_dynNewsHome+l_fileSep+l_htmlFile+".html",
         m_dynNewsEnv.m_dynNewsHome+l_fileSep+m_dynNewsEnv.m_batchGenXSL,
         m_dynNewsEnv.m_dynNewsHome+l_fileSep+l_xmlFile+".xml"
      );
      ...
    }
 ...
 }
```

The method `xmlnews.admin.AdminServlet.staticProcessingHtml` defines and executes a query to fetch the news items. Then it uses the Oracle XML-SQL Utility (XSU) to build an XML document from the result set and create an HTML page by applying an XSLT transformation.

```
public void staticProcessingHtml(String p_htmlFile,String p_xslfile,
   String p_xmlfile) throws Exception {
   String l_query = "select a.id, a.title, a.URL, a.DESCRIPTION, " +
   " to_char(a.ENTRY_DATE, 'DD-MON-YYYY'), a.CATEGORY_ID, b.name,
                  a.SUB_CATEGORY_ID, c.name, a.Type_Id, d.name, " +
```

```
" a.Submitted_By_Id, e.name, to_char(a.expiration_date, 'DD-MON-YYYY'),
                                                    a.approved_flag " +
" from news_items a, categories b, sub_categories c, types d, users e where " +
" a.category_id is not null and a.sub_category_id is not null and "+
" a.type_id is not null and a.EXPIRATION_DATE is not null and "+
" a.category_id = b.id  AND a.SUB_CATEGORY_ID = c.id AND a.Type_ID = d.id
                        AND " +
" a.SUBMITTED_BY_ID = e.id AND "+
" a.EXPIRATION_DATE > SYSDATE AND "+
" a.APPROVED_FLAG = \'A\' ORDER BY b.name, c.name ";

 Statement l_stmt = m_connection.createStatement();
 ResultSet l_result = l_stmt.executeQuery(l_query);
// Construct the XML Document using Oracle XML SQL Utility
   XMLDocument l_xmlDocument = m_xmlHandler.constructXMLDoc(l_result);
   l_stmt.close();

// Get the HTML String by applying corresponding XSL to XML.
   String l_htmlString = m_xmlHandler.applyXSLtoXML(l_xmlDocument,p_xslfile);

 File l_file = new File(p_htmlFile);
 FileOutputStream l_fileout = new FileOutputStream(l_file);
 FileOutputStream l_xmlfileout = new FileOutputStream(new File(p_xmlfile));.
 l_fileout.write(l_htmlString.getBytes());
 l_xmlDocument.print(l_xmlfileout);

 l_fileout.close();
 l_xmlfileout.close();
}
```

3 Combine News Items to Build a Document

The final step in personalizing content is converting XML documents into HTML pages according to end-user preferences.

The following code comes from `xmlnews.generation.SemiDynamicGenerate.dynamicProcessing`.

It retrieves the CLOBs corresponding to categories chosen by the user, converts each CLOB to an XML document, then combines them into one XML document. The process of converting the XML document to an HTML page is described in the next section.

```
public XMLDocument semiDynamicProcessingXML(Connection p_conn, UserPreference p_
prefs)
        throws Exception
 {
     String l_htmlString = null ;
     XMLDocument l_combinedXMLDocument = null ;
     XMLDocument[] l_XMLArray = new XMLDocument[p_prefs.m_categories.size()];
     int l_arrayIndex = 0 ;

     PreparedStatement l_selectStmt = p_conn.prepareStatement(
                 " SELECT PREGEN_XML FROM CATEGORIES_CLOB WHERE CATEGORY_ID =
?");
     // Process each preference.
     for ( ; l_arrayIndex < p_prefs.m_categories.size(); ++l_arrayIndex ){
       l_selectStmt.setString(1, p_prefs.m_categories.elementAt(l_
arrayIndex).toString());
       OracleResultSet l_selectRst = (OracleResultSet)l_
selectStmt.executeQuery();
       if (l_selectRst.next()) {
          CLOB l_clob = l_selectRst.getCLOB(1);
          l_XMLArray[l_arrayIndex] = convertFileToXML(l_clob.getAsciiStream());
       } else
          l_XMLArray[l_arrayIndex] = null ;
     }
     l_selectStmt.close();

     XMLDocHandler l_xmlHandler = new XMLDocHandler();
     l_combinedXMLDocument = l_xmlHandler.combineXMLDocunemts(l_XMLArray );
     return l_combinedXMLDocument ;
   }
```

4 Customizing Presentation

After fetching news items from the database, Dynamic News converts them to XML documents. XML separates content from presentation, making it easy to build custom HTML pages.

Dynamic News uses different XSL stylesheets to convert XML documents into HTML pages customized for various browsers:

- One for Netscape Navigator
- One for Microsoft Internet Explorer
- A generic stylesheet for other browsers.

It's a four-step process:

1. Get the user's browser type.
2. Get news items.
3. Build an XML document.
4. Convert XML to HTML.

Each time it receives an HTTP request, the application inspects the user-agent header to find out what kind of browser made the request. The following lines from xmlnews.dynamic.DynamicServlet.service show how the servlet creates a RequestHandler object (implemented in xmlnews/common/RequestHandler.java) and parses the request to get the browser type. Then the servlet uses this information to return an HTML page based on the end-user's preferences and browser type.

```
public void service(HttpServletRequest p_request, HttpServletResponse p_
response)
throws ServletException {
         ...
        // Instantiate a Request Handler (declared elsewhere)
         m_reqHandler = new RequestHandler(m_userPreference, m_general,m_
status);
         RequestParams l_reqParams = m_reqHandler.parseRequest(p_request, m_
connection);
         String l_browserType = l_reqParams.m_browserType;
         ...
         // Display the Dynamic Page
         this.sendDynamicPage(l_browserType,p_response,l_userName,m_
userPreference,
                              m_servletPath+"?REQUEST_TYPE=SET_ADVANCED_USER_
```

```
PREFS",
                m_servletPath+"?REQUEST_TYPE=LOGIN_REQUEST",
                m_servletPath+"?REQUEST_TYPE=LOG_OUT_REQUEST",
                m_servletPath);
          ...
     }
```

The code that actually extracts the browser type from the user-agent header resides in `xmlnews.common.GenUtility.getBrowserType`, which follows:

```
        public String getBrowserType(HttpServletRequest p_request) throws
Exception {

          // Get all the Header Names associated with the Request
          Enumeration l_enum = p_request.getHeaderNames();

          String l_Version     = null;
          String l_browValue   = null;
          String l_browserType = null;

          while (l_enum.hasMoreElements()) {
             String l_name = (String)l_enum.nextElement();
             if (l_name.equalsIgnoreCase("user-agent"))
                 l_browValue = p_request.getHeader(l_name);
          }

          // If the value contains a String "MSIE" then it is Internet Explorer
          if (l_browValue.indexOf("MSIE") > 0 ) {
              StringTokenizer l_st = new StringTokenizer(l_browValue, ";");
              // Parse the Header to get the browser version.
              l_browserType = "IE";
              while (l_st.hasMoreTokens()) {
                 String l_tempStr = l_st.nextToken();
                 if (l_tempStr.indexOf("MSIE") > 0 ) {
                    StringTokenizer l_st1 = new StringTokenizer(l_tempStr, " ");
                    l_st1.nextToken();
                    l_Version = l_st1.nextToken();
                 }
              }
          // If the value contains a String "en" then it is Netscape
          } else if (l_browValue.indexOf("en") > 0) {
              l_browserType = "NET";
              String l_tVersion = l_browValue.substring(8);
              int l_tempInd  = l_tVersion.indexOf("[");
              l_Version = l_tVersion.substring(0, l_tempInd);
```

```
        }

        // Return the Browser Type and Version after concatenating
        return l_browserType + l_Version;
      }
```

After getting the end-user's browser type, the DynamicServlet's service method passes it to `xmlnews.dynamic.DynamicServlet.sendDynamicPage`.

This method generates HTML by fetching XML documents from the database and converting them to HTML by applying an XSL stylesheet appropriate for the end-user's browser type.

```
public void sendDynamicPage(String p_browserType,HttpServletResponse p_response,
  String p_userName,UserPreference p_pref,String p_userPrefURL,
  String p_signOnURL,String p_logout,
  String p_servletPath) throws Exception {
  String l_finalHTML = ""; // Holds the html
     if (p_browserType.startsWith("IE4") || (p_browserType.startsWith("IE5"))) {
         // Send the XML and XSL as parameters to get the HTML string.
         l_finalHTML = m_handler.applyXSLtoXML(
                             this.dynamicProcessingXML(m_connection, p_pref),
                             m_dyEnv.m_dynNewsHome + "/DynamicIE.xsl"
                             );
  String l_thisbit = m_general.postProcessing(l_finalHTML,p_userName,
         p_userPrefURL,p_signOnURL,p_logout,p_servletPath);
         PrintWriter l_output = p_response.getWriter();
         l_output.print(l_thisbit);
         l_output.close();
     }
      else if (p_browserType.startsWith("NET4") ||
         (p_browserType.startsWith("NET5"))) {
         // Do the same thing, but apply the stylesheet "/DynamicNS.xsl"
         ...
      // When the Browser is other than IE or Netscape.
      } else {
      // Do the same thing, but apply the stylesheet "/Dynamic.xsl"
      ...
      }
   }
```

The key methods are:

- `xmlnews.dynamic.DynamicServlet.dynamicProcessingXML`

This queries the database for news items that match the end-user's preferences. It converts the result set into an XML document by calling `xmlnews.common.XMLDocHandler.constructXMLDoc`.

- `xmlnews.common.XMLDocHandler.applyXSLtoXML`

  This converts an XML document into HTML using a specified XSL stylesheet. It uses XSL Transformation capabilities of Oracle XML Parser Version 2.0. More specifically, it uses the Document Object Model (DOM) parser to create a tree that represents the structure of the XML document. To build the final HTML string, it creates an element to serve as the root of the tree, then appends the parsed DOM document.

Importing and Exporting News Items

Dynamic News can also import and export XML documents that conform to the Resource description framework Site Summary (RSS) standard. Developed by Netscape as a way to share data channels, RSS is used at Web sites such as my.netscape.com and slashdot.org.

An application can use RSS to syndicate its news pages (making them available to RSS hosts) and to aggregate news from other RSS sites. For example, Dynamic News includes the `xmlnews.admin.RSSHandler` class. It uses a specified DTD to parse and extract news items from a specified file, and then it stores the items in a Hashtable. The class also provides a method that returns the elements in that Hashtable.

7

Personalizing Data Display With XML: Portal-to-Go

This chapter contains the following sections:

Introduction to Oracle Portal-to-Go

Most Web clients are PCs, but according to the Meta Group, “By 2003, over 50% of internet access will be by non-PCs.”

Oracle Portal-to-Go (Portal-to-Go) enables the following services:

- It allows virtually any wireless device to access any existing Web or database application or content, including secure e-business applications.
- It enables wireless carriers to become broad-range e-commerce service providers.

Portal-to-Go, a component of the Oracle Internet Platform, is a server product that provides everything you need to deliver Web content to any capable device. It transforms existing content to a device's native format, and provides a portal interface for the end-user.

XML is the Key

XML is the key for content providers to reach an audience of mobile users with data delivered in many different formats. XML isolates the *source* content format from the *target* device format, enabling content providers to take data from any source and deliver it to any target. Use these XML-based techniques in applications that convert data from one format to another, such as:

- Enterprise application integration
- Customization of content delivery based on user profile
- Content and services aggregation in the form of a marketplace supplier exchanges

Portal-to-Go Components

A Portal-to-Go portal includes the following components:

- Services that deliver data to mobile devices
- Adapters that convert HTML and RDBMS content to XML
- Transformers that convert XML to the appropriate markup language

This chapter describes how Portal-to-Go uses XML to make Web content available to any device. It describes a stock quote service and the role XML takes as an intermediate format for the data exchange.

Oracle XML Components

The XML Parser for Java, V2 is used in the Portal-to-Go Adapters and Transformers. The XSL-T package of the XML Parser for Java is also used.

Portal-To-Go 1.0.2 Features

The Portal-To-Go 1.0.2 features include the following:

- Apache and Apache JServ Support
- Explicit content-type setting on a per service basis
- Improved customization of device output, such as explicit settings of output variable names
- Support for optional input parameters
- Improved handling of multibyte character sets
- Improved handling of special characters, such as "$"
- Enhancements to existing WML transformers
- A single WML transformer replaces a number of separate transformers from previous releases
- Bookmarks functionality to allow external sites to be included within a Portal-to-Go portal
- Improvements to allow the creation of portals using multibyte character sets

> **See Also:** the *Portal-to-Go Installation Guide* for more details on repository upgrades.

What's Needed to Run Portal-to-Go

Portal-to-Go requires the following:

- Oracle8*i*, Release 8.1.5 or above
- One of the following servers:
  - *i*AS (Internet Application Server)
  - Apache Web Server 1.3.9 and Apache JServ 1.1
- Java Configuration Requirements

- Service Designer. The Portal-to-Go Service Designer requires JDK 1.2.2. You can install JDK 1.2.2 from the Portal-to-Go CD-ROM.
- Web Integration Developer. The Web Integration Developer includes its own Java Virtual Machine (JVM). It does not require any Java setup.
- Server Component. The Portal-to-Go server component runs with JDK 1.1 or 1.2. JDK 1.2 has improved performance.

Portal-To-Go: Supported Devices and Gateways

Transformers

Portal-to-Go provides transformers for the latest WAP-compliant devices from the following vendors:

- Alcatel
- Ericsson (including R320)
- Motorola (including Timeport)
- Neopoint (including NP1000)
- Nokia (including 7100)
- Samsung

You can also create your own transformers and extend Portal-to-Go support to other devices.

WAP Gateways

Portal-to-Go has been successfully tested with the following WAP gateways:

- Phone.com UP.link Gateway
- Nokia WAP Gateway
- Ericsson WAP Gateway
- Infinite Technologies WAPLite

How Portal-to-Go Works

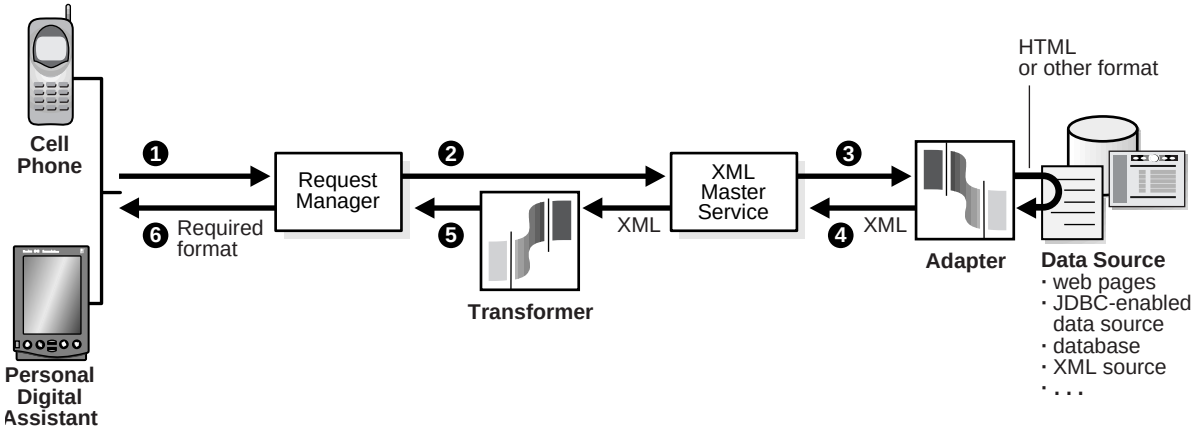
Figure 7–1 shows how Portal-to-Go works. When an end-user requests a Portal-to-Go service, the following actions transpire:

1. Portal-to-Go's Request Manager performs user-level preprocessing, including authentication.
2. Request Manager sends a request to the corresponding Master Service.
3. Master Service invokes an adapter to retrieve the requested content.
4. Adapter returns the content in XML.
5. Transformer converts the XML content into a format appropriate for the target device.
6. Request Manager returns the information to the device.

XML and related technologies are at the core of Portal-to-Go's functionality as follows:

- XML separates presentation and content
- A DTD maps XML tags to User Interface (UI) elements
- XSL stylesheets define rules for formatting, sorting, and filtering results

Figure 7–1 How Portal-to-Go Works



Portal-to-Go Components

Portal-to-Go Services

A Portal-to-Go service encapsulates a unit of information requested by, and delivered to, a Portal-to-Go client. Examples of services include:

- Stock quotes
- News
- Maps
- Email

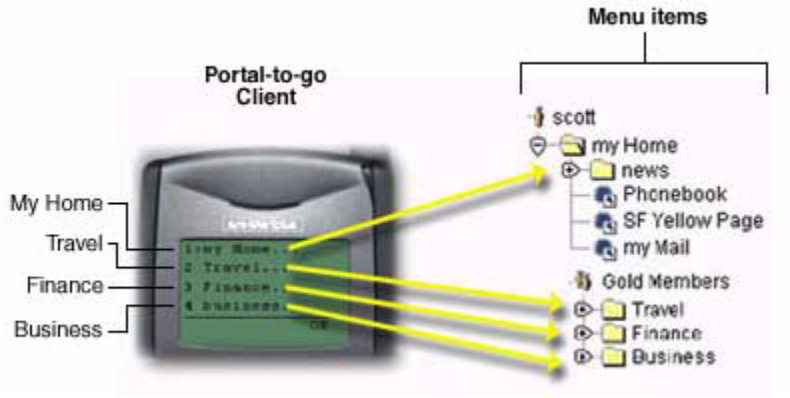
You can build services from an existing Web site, a query to any database, or any XML source.

Master Service

A Master Service is a Portal-to-Go object that implements a service and invokes a specific adapter. The end-user typically sees a service as a menu item on a handset or a link on a Web page. End-users invoke Master Services by choosing menu items in their device interface. The Master Service returns the following kinds of data:

- Static text, such as a movie review
- An application, such as an airline booking system

Figure 7–2 How an End-User Sees Services as Menu Items. Master Service is Invoked When You Select a Menu Item



By mapping an Adapter to device Transformers, master services link Portal-to-Go content sources to the delivery platforms. Each Master Service is based on one Adapter. A Master Service creates its own instance of the Adapter it uses. Therefore, several services can use the same type of Adapter, and each can pass the Adapter its service-specific argument values.

Portal-to-Go Adapters

A Portal-to-Go Adapter is a Java application that retrieves data from an external source and renders it in Portal-to-Go XML. When invoked by a Master Service, an Adapter returns an XML document that contains the service content. Adapters provide the interface between the Portal-to-Go server and the content source.

An Adapter does the following:

- Connects to a data source
- Retrieves content
- Converts the content to Portal-to-Go XML

Portal-to-Go provides pre-built Adapters for popular content sources, including Web pages and JDBC-enabled data sources, and adapters you can modify to work with other content sources.

All adapters must generate Portal-to-Go XML. This is a well-formed, valid XML document that complies with the Portal-to-Go DTD.

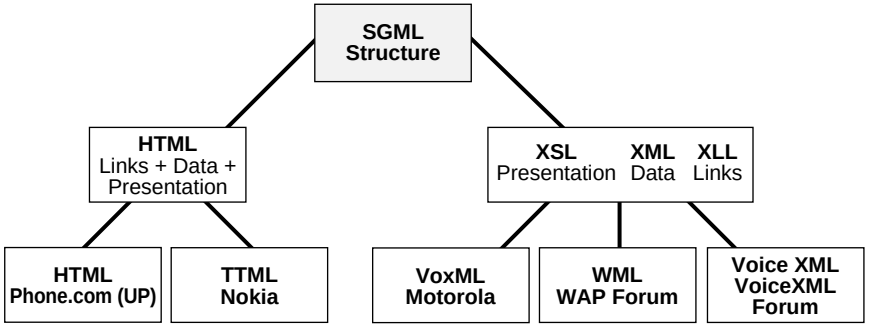
Portal-to-Go Transformers

Portal-to-Go Transformers are Java programs or XSL-T stylesheets that convert an XML document into the target or another Portal-to-Go format. They can also rearrange, filter, and add text. The Transformers enable you to present content in a format best suited to your target device. Portal-to-Go supplies transformers for the following markup languages:

- WML 1.1 - The wireless markup language defined by the WAP Forum.
- Tiny HTML - A subset of HTML, suitable for handheld devices (not phones) such as Palm Pilots.
- VoxML - The Motorola markup language that enables voice interaction with applications.
- TTML - The Tagged Text Mark-up Language is a subset of HTML developed by Nokia.
- HDML - The Handheld Devices Markup Language is a simplified version of HTML designed specifically for handheld devices.
- Plain Text - Converts content for Short Message Service-capable devices and email applications.

Figure 7–3 illustrates these markup languages and their derivation.

Figure 7–3 Portal-to-Go Supports Several HTML- and XML-based Markup Languages



Use Transformers to optimize content presentation for any device, and support new device platforms. In most cases, you can simply modify or re-use an existing Transformer.

Exchanging Data via XML: Source to XML, XML to Target with Portal-to-Go

With XML as an intermediate format, you can take data from any source and deliver it to any device. Suppose you have a Web application that provides stock quotes and headlines, and you want to deliver the information to a mobile phone and a PDA (Personal Digital Assistant, such as a Palm Pilot).

Because each device has specific requirements for formatting content, you cannot send the same data to each device. How would you do it? Portal-to-Go defines an intermediate data format in XML. It also provides tools that allow content providers to perform the following tasks:

- Extract source content
- Convert source content to XML
- Transform XML to the markup language for each device

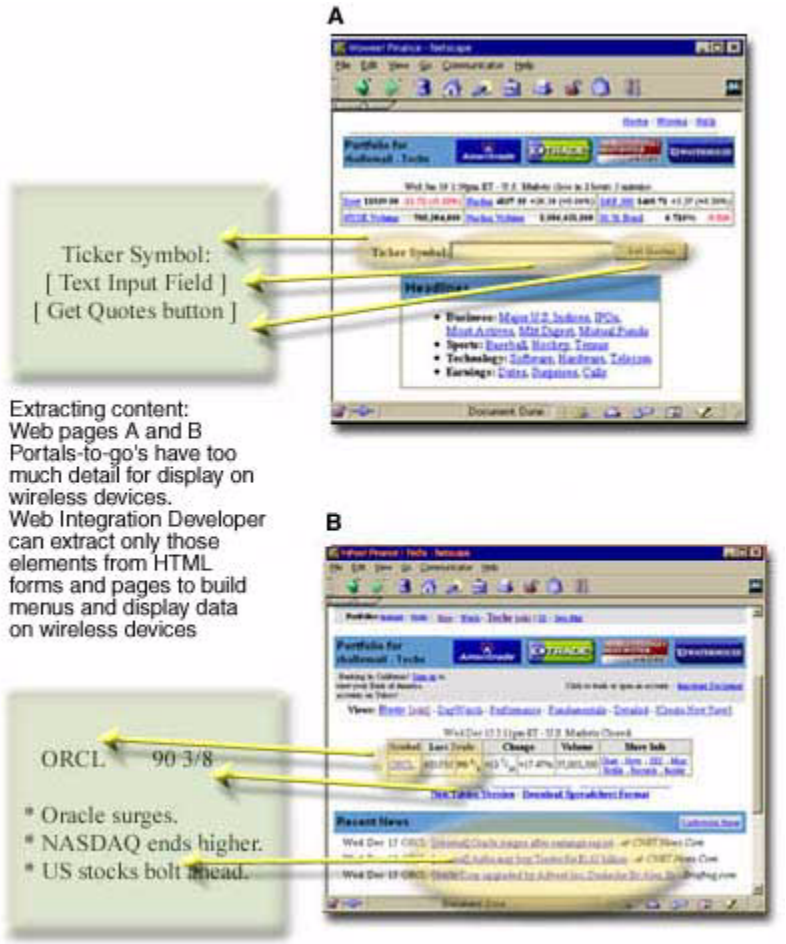
Extracting Content

Hand-held devices cannot display as much information as a desktop monitor, so you have to be selective. Figure 7–4 shows two, deliberately undecipherable, Web pages from a Stock Data application.

- A — The first page is a form where you enter a company's ticker symbol. For example, ORCL is the ticker symbol for Oracle Corporation.
- B — The second page displays the stock price, and other information about the company.

Both pages are full of ads, buttons, hyperlinks, related articles, and more. Your first step would be to identify the elements of a Web page that you want to make accessible to your service.

Figure 7–4 Extracting Elements from HTML Pages For Display on Wireless Devices



Web Integration Developer: A "Screen Scraper"

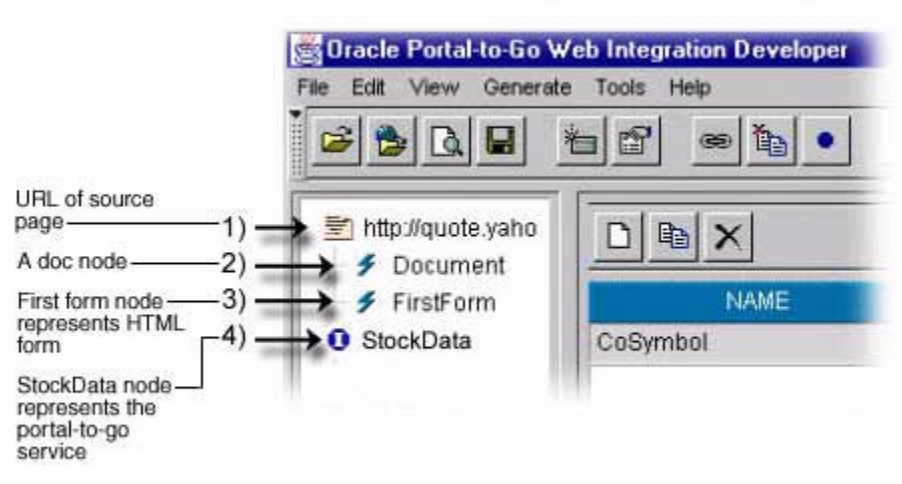
Portal-to-Go provides a GUI tool called the Web Integration Developer. It's a "screen scraper" that *extracts* user interface (UI) elements from a Web page. Using Web Integration Developer tools and functions, you choose UI elements and define corresponding output and input parameters.

Figure 7–5 shows a corner of Web Integration Developer. The Document Browser panel includes the following items:

- The URL of the source page
- A Document node that represents the contents of the source page: paragraphs, images, links, lists, and tables
- A FirstForm node that represents an HTML form. The Web Integration Developer creates a form node for each form in the source page.
- A StockData node that represents the Portal-to-Go service.

The next step is to convert the extracted elements to XML.

Figure 7–5 Using Web Integration Developer to "Scrape the Screen"



Converting to XML

A Portal-to-Go Adapter retrieves content from the source. In the example illustrated here, it pulls specific quotes and headlines from a Web page. Then the Adapter converts the content to XML.

Why Use an Intermediate XML Format?

Why not go straight to the target device format? Two reasons: flexibility and extensibility. To go straight from source to target, you must effectively create an adapter and transformer for each source-target pair. With XML as an *intermediate format*, you only need one adapter for each source, and one transformer for each device. For example if there are, say two content sources and three target devices

- *Source to Target, without XML*: You will need six adapter-transformer pairs, namely *twelve* components altogether
- *Source to Target, with XML*: You will need only *five* components altogether, two adapters and three transformers.

Using the Simple Result DTD

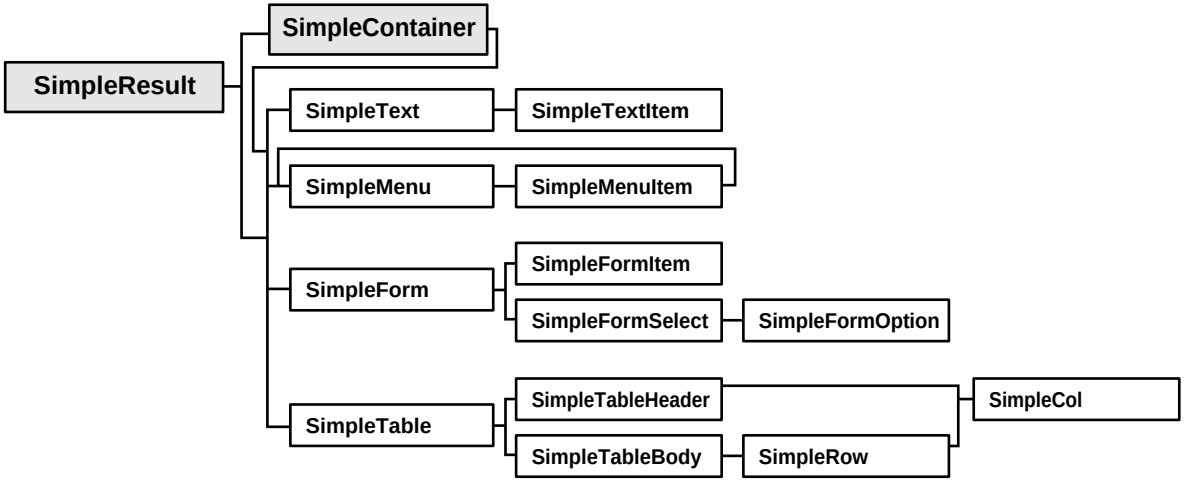
Adapter output must be XML to be generic. The key is to define an XML document type that can represent any data type you might want to display on any device. The document type is defined by a Document Type Definition (DTD). A DTD is a file that provides a grammar for a class of XML documents by describing the elements it can contain.

To create a truly universal intermediate data format, Portal-to-Go uses the Simple Result DTD. Elements in the Simple Result DTD represent the elements of an abstract user interface. These include the following:

- Text items
- Menus
- Forms
- Tables

Figure 7–6 illustrates the Simple Result DTD content model.

Figure 7–6 Simple Result DTD Content Model



Following is a portion of SimpleResult.dtd that shows the elements used in our Stock Data example.

```
<!--
Entity:   "GENATTR" contains generic attributes for most elements.
Attribs:  "name" is the name of the element.
          "title" is the title of the element.
...
-->
<!ENTITY % GENATTR "
            name    CDATA #IMPLIED
            title   CDATA #IMPLIED
 ...        ">

<!--
Element:    "SimpleResult" is the result element.
Usage:      This element contains the result.
Children:   "SimpleText" is a text result.
 ...
-->

<!ELEMENT   SimpleResult ((SimpleContainer|SimpleText|SimpleMenu|
            SimpleForm|SimpleTable|SimpleImage|SimpleBreak)+)>
<!ATTLIST   SimpleResult %GENATTR;>
```

```
...

<!--
Element:    "SimpleText" for displaying one or more blocks of text.
Usage:      Used for plain text.
Children:   "SimpleTextItem" is a block of text.
-->

<!ELEMENT   SimpleText (SimpleTextItem+)>
<!ATTLIST   SimpleText %GENATTR;>

<!--
Element:    "SimpleTextItem" is a block of text
Usage:      Holds one block of text - normally a single paragraph.
Children:   "#PCDATA" is the actual text.
-->

<!ELEMENT   SimpleTextItem (#PCDATA)>
<!ATTLIST   SimpleTextItem %GENATTR;>

...

<!--
Element:    "SimpleForm" for displaying one or more input fields.
Usage:      As a data-entry form.
Children:   "SimpleFormItem" for each input field.
Attribs:    "target" is the link target for this form.
            "section" is the section identifier

***** A special case for the WIDL adapter *****

-->
<!ELEMENT    SimpleForm ((SimpleFormItem|SimpleFormSelect)+)>
<!ATTLIST    SimpleForm %GENATTR;
             target  CDATA #REQUIRED
             section CDATA #IMPLIED>
<!--
Element:    "SimpleFormItem" is a single input item in a simple form.
Usage:      For getting input from a user.
Children:   "#PCDATA" contains pre-filled input from the server.

****** This overrides the default attribute. *******

Attribs:    "default" provides a default value for optional fields.
```

***** The default value should only be used if the field is empty.

"mandatory" indicates that the form item is mandatory.

"maxLength" provides a maximum input length.

```
-->
<!ELEMENT   SimpleFormItem (#PCDATA)>
<!ATTLIST   SimpleFormItem %GENATTR;
            default  CDATA #IMPLIED
            mandatory(yes|no) "no"
            maxLength CDATA #IMPLIED>

...
```

Adapters Map the Source Content to the DTD Element

Portal-to-Go Adapters map the source content to the appropriate Simple Result element.

- *Input bindings* specify any data required to complete the request through form <input> tags and variables in the service URL.
- *Output bindings* are the results returned to the requester. They select only the relevant pieces of HTML for the service and device.

For example, Table 7–1 shows the XML for an input form (text label, input field, and submit button) and results page (ticker symbol, stock price, and headlines) generated by a hypothetical StockData Adapter.

Table 7–1 XML for Input and Results Page Generated by StockData Adapter

XML for Input Page	XML Results Page: Quote and Headlines Page
<SimpleResult> <SimpleText> <SimpleTextItem name = "TickerField"> Ticker Symbol: </SimpleTextItem> </SimpleText> <SimpleForm title="Input Form"> <SimpleFormItem name="Ticker"> </SimpleFormItem> <SimpleFormButton name="submitBtn"> Get Quote </SimpleFormButton> </SimpleForm> </SimpleResult>	<SimpleResult> <SimpleText title="Quote Results"> <SimpleTextItem name="Ticker"> ORCL </SimpleTextItem> <SimpleTextItem name="Price"> 90 3/8 </SimpleTextItem> </SimpleText> <SimpleText title="Headlines"> <SimpleTextItem name = "Headline1"> * Oracle surges. </SimpleTextItem> <SimpleTextItem name = "Headline2"> * NASDAQ closes higher. </SimpleTextItem> <SimpleTextItem name = "Headline3"> * US stocks bolt ahead. </SimpleTextItem> </SimpleText> </SimpleResult>

Sample Adapter Classes

The following two code examples show how Adapters are implemented in Java. Study them to learn how Adapters work. You can modify them to create your own Adapters for custom content sources.

- "Portal-to-Go Adapter Example 1: Converts Stock Quotes and Headlines to XML" shows how an Adapter class converts stock quotes and headlines to XML. For clarity, auxiliary methods are omitted.
- "Portal-to-Go Adapter Example 2: Greets Users by Name" is a simple, but complete, Adapter implementation that greets users by name.

Portal-to-Go Adapter Example 1: Converts Stock Quotes and Headlines to XML

Consider an Adapter class that must implement several methods. The key method is `invoke`. Master Service calls `invoke` every time a client makes a request. This example illustrates the Adapter class `invoke` method that generates the XML result for the quote and headlines page shown in Table 7–1.

```
public class StockQuoteAdapter implements Adapter {
  ...
  public Element invoke (ServiceRequest sr)
    throws AdapterException {
      Element result = XML.makeElement("SimpleResult");
      result.setAttribute("title", "Stock Data");

      Element quote = XML.makeElement("SimpleText");
      quote.setAttribute("title", "Quote");

      Element tickerSymbol = XML.makeElement("SimpleTextItem");
      tickerSymbol.setAttribute ("title", "Ticker");
      String t = sr.getArguments().getInputValue("Ticker");
      Text ticker = XML.makeText(t);
      tickerSymbol.appendChild(ticker);
      quote.appendChild(tickerSymbol);

      Element stockPrice = XML.makeElement("SimpleTextItem");
      tickerSymbol.setAttribute ("title", "Price");
      String p = sr.getArguments().getInputValue("Price");
      Text price = XML.makeText(p);
      stockPrice.appendChild(price);
      quote.appendChild(stockPrice);
      result.appendChild(quote)
```

```
      Element headlines = XML.makeElement("SimpleText");
      headlines.setAttribute ("title", "Headlines");
      Element headline = XML.makeElement("SimpleTextItem");
      int i = 0;
      String argBase = "headline";
      String h = "";
      while (h != null) {
         h = sr.getArguments().getInputValue(argBase + i);
         headline.setAttribute("name", argbase + i);
         Text headText = XML.makeText(h);
         headline.appendChild(headText);
         headlines.appendChild(headline);
         i++;
      }
      result.appendChild(headlines);
      return result;
  }
  ...
}
```

Portal-to-Go Adapter Example 2: Greets Users by Name

Consider a simple Adapter for a service that greets users by name. It has the following inputs:

- An initialization parameter, the string used for the greeting
- An input parameter, the name of the user

Example 2's Adapter uses the `invoke` method to build a Simple Result document using methods in the following packages:

- `org.w3c.dom.Element`
- `org.w3c.dom.Text`

The invoke method performs the following tasks:

1. Creates the root result element
2. Creates a SimpleText element. Sets its title attribute, and appends the element to the root element. As defined in the Simple Result DTD, a SimpleTextItem is a required child element of SimpleText.
3. Retrieves the input parameter value, appends it to the result document
4. Returns the result

Here is the Adapter implementation:

```
import org.w3c.dom.Element;
import org.w3c.dom.Text;
import oracle.panama.Argument;
import oracle.panama.Arguments;
import oracle.panama.ServiceRequest;
import oracle.panama.adapter.Adapter;
import oracle.panama.adapter.AdapterDefinition;
import oracle.panama.adapter.AdapterException;
import oracle.panama.adapter.AdapterHelper;

public class HelloAdapter implements Adapter {
   private boolean initialized = false;
   private String greeting = "Hello";
   public static final String GREETING = "greeting";
   public static final String NAME = "name";

   // Called once, when the adapter is instantiated.
   public void init (Arguments args) throws AdapterException {
      synchronized (this) {
         if(!initialized) {
            initialized = true;
            greeting = args.getInputValue( GREETING );
         }
      }
   }
  public Element invoke (ServiceRequest sr)
                          throws AdapterException {
      Element result = XML.makeElement("SimpleResult");
      Element st = XML.makeElement("SimpleText");
      st.setAttribute ("title",
                  "Oracle Portal-to-Go Server HelloAdapter Sample");
      result.appendChild (st);
      Element sti = XML.makeElement("SimpleTextItem");
      sti.setAttribute ("name", "message");
      sti.setAttribute ("title", "Portal-to-Go says:");
      st.appendChild (sti);
      // ServiceRequest sr contains input parameters (like NAME, below).
      String name = sr.getArguments().getInputValue(NAME);
      Text txt = XML.makeText( greeting + " " + name + "!");
      sti.appendChild (txt);
      return result;
   }
   // This method enables master services to determine
```

```
   // the initialization parameters used by the adapter.
   private AdapterDefinition initDef = null;
   public AdapterDefinition getInitDefinition() {
      if (initDef == null) {
         synchronized (this) {
            if (initDef == null) {
               initDef = AdapterHelper.createAdapterDefinition();
               initDef.createInit( Argument.SINGLE_LINE,
                                   GREETING,
                                   "Greeting phrase",
                                   null );
             }
         }
      }
      return initDef;
   }
   // This method defines the adapter&#146;s runtime input parameters.
   private AdapterDefinition adpDef = null;
   public AdapterDefinition getAdapterDefinition() throws AdapterException {
      if (adpDef == null ) {
         synchronized (this) {
            if (adpDef == null) {
                if (initDef == null)
                   throw new AdapterException ("Adapter is
                                   not properly initialized");
                adpDef = initDef;
                adpDef.createInput( Argument.SINGLE_LINE,
                                    NAME,
                                    "Name to greet",
                                    null );
             }
         }
      }
      return adpDef;
   }
}
```

When invoked with an input parameter of "Dolly", the above Adapter returns the following XML result:

```
<SimpleResult>
   <SimpleText title="Oracle Portal-to-Go Server Hello Sample">
      <SimpleTextItem name="message" title="Portal-to-Go says:">
         Hello Dolly!
      </SimpleTextItem>
```

```
   </SimpleText>
</SimpleResult>
```

Transforming XML to the Target Markup Language

Portal-to-Go Transformers convert XML documents into the markup language for the target device. By using a generic internal XML format, such as SimpleResult, to represent information, you can take full advantage of each client device's UI capabilities.

The Transformers use the SimpleResult DTD to map abstract UI elements to the target format. You can implement a Transformer using Java or XSL-T, depending on what you need to do:

- *Java*. Java lets you add device-specific behavior, such as a Repeat function for a VOX device, which isn't needed for a device that writes to the screen. See "Portal-to-Go Java Transformer Example 1: Converting Simple Result Elements to Another Format".
- *XSL -T*. XSL Style sheets can include complex pattern matching and result handling logic. They typically include literal result elements, such as the target format markup tags. Portal-to-Go uses XSL style sheets by default. See "Portal-to-Go XSL Stylesheet Transformer Example 1: Converting Simple Result Documents to Plain Text".

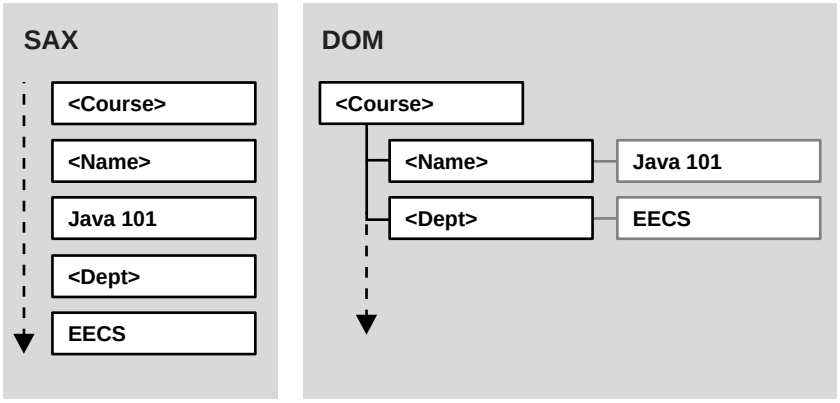
Portal-to-Go: Java Transformers

You can create Java Transformers using either of the following two interfaces:

- Document Object Model (DOM) interface, which manipulates the tree-based document object model
- Simple API for XML (SAX) interface, which interacts directly with events in the parsing process.

These two interfaces are illustrated in Figure 7–7.

Figure 7–7 The DOM and SAX Interfaces



Portal-to-Go includes a Java Transformer that converts Simple Result documents to plain text. The Transformer does not create markup tags in the resulting document, but it does apply simple text formatting elements, such as line breaks and tabs.

Portal-to-Go Java Transformer Example 1: Converting Simple Result Elements to Another Format

Though simple, this example shows how you can convert Simple Result elements into another format.

```
package oracle.panama.core.xform;
import org.w3c.dom.NodeList;
import org.w3c.dom.Element;
import oracle.panama.PanamaException;
import oracle.panama.core.LogicalDevice;
import oracle.panama.core.Service;
```

```
import oracle.panama.Arguments;
import oracle.panama.core.parm.PanamaRequest;
import oracle.panama.core.parm.AbstractRequest;

public class SimpleResultToText implements Transform {
    public SimpleResultToText() {}

    private String format(Element el) {
        if (el == null) {
            return "";
        }
        StringBuffer buf = new StringBuffer();
        String name = el.getTagName();
        if (name != null && name.length() > 0) {
            buf.append(name);
            buf.append(": ");
        }
        buf.append(el.getNodeValue());
        return buf.toString();
    }

    public String transform(Element element, LogicalDevice device)
                                    throws PanamaException {
        PanamaRequest req = AbstractRequest.getCurrentRequest();
        Service service = req.getService();
        StringBuffer buf =
          new StringBuffer((service == null) ? "" : service.getName());
        NodeList list = element.getElementsByTagName("*");
        Element el;
        String tag;
        boolean newRow = false;
        for (int i = 0; i
            el = (Element)list.item(i);
            tag = el.getTagName();
            if (tag.equals("SimpleRow")) {
                newRow = true;
                buf.append("\n");
            } else if (tag.equals("SimpleCol")) {
                if (!newRow) {
                    buf.append("\t");
                } else {
                    newRow = false;
                }
                buf.append(format(el));
            } else if (tag.equals("SimpleText") ||
```

```
                        tag.equals("SimpleForm") ||
                        tag.equals("SimpleMenu")) {
                    newRow = true;
                    buf.append("\n");
                } else if (tag.equals("SimpleTextItem") ||
                        tag.equals("SimpleFormItem") ||
                        tag.equals("SimpleMenuItem")) {
                    if (!newRow) {
                        buf.append("\n");
                    } else {
                        newRow = false;
                    }
                    buf.append(format(el));
                }
            }
            return buf.toString();
        }
    }
```

Portal-to-Go: XSL Stylesheet Transformers

XSL stylesheets are XML documents that specify the processing rules for other XML documents. An XSL stylesheet, like a Java Transformer, is specific to a particular DTD, and should handle all elements declared in that DTD. When it finds an element in a source document, it follows the rules defined for the element to format its content.

Portal-to-Go XSL Stylesheet Transformer Example 1: Converting Simple Result Documents to Plain Text

This XSL Transformer example is included in the Portal-to-Go initial repository and is the XSL version of the Java Transformer shown above. It converts Simple Result documents to plain text.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">
   <xsl:template match="/">
      <xsl:apply-templates></xsl:apply-templates>
   </xsl:template>
      <xsl:template match="SimpleTextItem | SimpleFormItem | SimpleMenuItem">
      <xsl:text>
      </xsl:text>
      <xsl:value-of select="."></xsl:value-of>
   </xsl:template>
   <xsl:template match="SimpleRow">
      <xsl:text></xsl:text>
      <xsl:for-each select="./SimpleCol">
         <xsl:text></xsl:text>
         <xsl:value-of select="."></xsl:value-of>
      </xsl:for-each>
   </xsl:template>
</xsl:stylesheet>
```

In this example. the XSL stylesheet performs the following tasks:

1. Selects a Simple Result element using pattern-matching semantics. The element "/", for example, matches the document's root element.
2. Uses `apply-templates` to process the contents of that element.
3. Descends the source element tree, selecting and processing each sub-element. Character instructions, such as `value-of` and `for-each`, manipulate the content of matching elements.
   - The `value-of` element extracts the actual content of the element.

- The `for-each` element applies iterative processing.

Each Markup Language Requires a Unique Transformer

Each unique markup language requires a unique Transformer. The Stock Data example assumes that the PDA and cell phone use different markup languages (Tiny HTML and WML), so we need two Transformers. Once they're built, though, these Transformers can process content from any Adapter that generates Simple Result XML.

Table 7–2 lists the Adapter's SimpleResult XML code and the markup language generated by two transformers:

- Tiny HTML for the PDA, which can format and display both quotes and headlines
- WML for the cell phone, which can only display quotes.

Table 7–2 Using Unique Transformers to Transform the Adapter's Simple Result XML

Adapter's Simple Result XML	Unique Transformers
<SimpleResult> <SimpleText title="Quote"> <SimpleTextItem name="Ticker"> ORCL </SimpleTextItem> <SimpleTextItem name="Price"> 90 3/8 </SimpleTextItem> </SimpleText> <SimpleText title="Headlines"> <SimpleTextItem name = "Headline1"> * Oracle surges. </SimpleTextItem> <SimpleTextItem name = "Headline2"> * NASDAQ closes higher. </SimpleTextItem> <SimpleTextItem name = "Headline3"> * US stocks bolt ahead. </SimpleTextItem> </SimpleText> </SimpleResult>	**Tiny HTML for PDA** <html> Quote Ticker: ORCL Price: 90 3/8 Headlines: * Oracle surges. * NASDAQ closes higher. * US stocks bolt ahead. </html> **WML for Cell Phone** <?xml version "1.0"?> <!DOCTYPE WML PUBLIC "-//WAPFORUM/DTD WML 1.0//EN" "http://www.wapforum.org.DTD.wml.xml"> <WML> <CARD NAME="QUOTE_CARD" TITLE="Quote Card"> ORCL 90 3/8 </CARD> </WML>

Portal-To-Go Stylesheet Transformer Example 2: Customizing a WML1.1 Transformer Stylesheet

WML Browsing on Phone.com Browsers

When using the Phone.com browser the navigation model requires you to select the [Link] option before proceeding. You can customize the stylesheet to change this behavior. For example, you can add the following to the WML1.1 Transformer stylesheet:

```
| The SimpleForm Mapping
+-->
<xsl:template match="SimpleForm">
<p>
 <xsl:variable name="theTarget">
 <xsl:value-of select="@target"/>
 <xsl:for-each select="SimpleFormItem | SimpleFormSelect">
 <xsl:text>&#38;</xsl:text>
 <xsl:value-of select="@name"/>
 <xsl:text>=$(</xsl:text>
 <xsl:value-of select="@name"/>
 <xsl:text>)</xsl:text>
</xsl:for-each>
</xsl:variable>
<xsl:apply-templates/>

<!-- Ensure [Link] is selected -->
<select>
<option>
<onevent type="onpick">
<go href="{$theTarget}"/>
</onevent>
<xsl:choose>
<xsl:when test="boolean(@submit)">
<xsl:value-of select="@submit"/>
</xsl:when>
<xsl:otherwise>Submit</xsl:otherwise>
</xsl:choose>
</option>
</select>
</p>

<!-- Ensure [Link] is selected ends -->
<!--
```

```
<a href="{$theTarget}">
  <xsl:choose>
    <xsl:when test="boolean(@submit)">
    <xsl:value-of select="@submit"/>
    </xsl:when>
  <xsl:otherwise>Submit</xsl:otherwise>
  </xsl:choose>
</a>
<br/>
</p>
-->
</xsl:template>
```

Portal-to-Go Case Study 1: Extending Online Drugstore's Reach

An online drugstore is using Oracle® Portal-to-Go wireless Internet software to extend its reach to customers, providing convenience and around-the-clock access to its online drugstore through hand-held devices.

Oracle Portal-to-Go extends the existing Internet site to hand-held wireless devices. In this case Portal-to-go integrates with the online store, which is built on Oracle Internet Platform. The solution allows consumers to purchase the full line of drugstore products from virtually anywhere.

Portal-to-Go renders any Internet content devices independent, hence allowing existing content designed for PCs to be made accessible from virtually any device connected to the Internet, such as personal digital assistants (PDAs), wireless application protocol (WAP) phones, or even pagers.

Portal-to-Go Case Study 2: Expanding Bank Services

A bank is now offering online services to its customers through mobile phones and uses the Oracle wireless Internet server product, Oracle® Portal-to-Go.

The bank's customers have access to financial quotes, a search facility for finding the nearest branch office, a loan repayment calculator, an events calendar, and weather reports from either their WAP (wireless application protocol)-enabled phones, or standard GSM phones.

The bank is also adding transactional banking services to their wireless Internet offering. With this, the bank's new WAP platform will also allow access to the bank's online information and services through customer mobile phones.

8

Customizing Presentation with XML and XSQL: Flight Finder

This chapter contains the following sections:

XML Flight Finder Sample Application: Introduction

XML Flight Finder fetches data about airline flights and customizes the results for the client device (PC, cell phone, PDA,...). It is built on Oracle8*i* and leverages Oracle XSQL Servlet, hence this application can submit SQL queries and define output formats using XML, XSL, and XSQL text files — no Java programming is required, and there is no code to compile. This application is easy to build, customize, and maintain.

Download the source code for XML Flight Finder to study and modify. You can also read an article that describes how the Flight Finder uses Oracle XML products and technologies, and there's a page of links to sites where you can download software that lets you simulate, for example, a cell phone on your PC.

This information and the application download is also available at:

http://technet.oracle.com/sample_code/index.htm

http://technet.oracle.com/tech/xml/xsql_servlet/index.htm then select Sample Code

Required Software

To build and run the XML Flight Finder application you need the following:

- Java 1.2 or higher.
- Oracle8i 8.1.5 or higher.
- A version of SQL*Plus compatible with your database.
- Oracle XSQL Servlet (includes Web-to-Go personal Web server for Windows NT). Download the latest version from OTN.
- Flight Finder files. Download fly.zip.
- A Web browser. For best results, use one that can process XML (such as Internet Explorer 5).
- (Optional) Software that simulates other devices (such as a cell phone) on a computer.
- (Optional) Apache or iAS Web Server. While Web-to-Go is all you need to run the Flight Finder on your own machine under Windows NT, you can also run the Flight Finder under Apache or iAS.

How Flight Finder Works

Flight Finder queries the database for information about flights from one city to another, then returns the results in a format customized for your end-user's device. Built on Oracle8*i*, Flight Finder uses the following products and technologies:

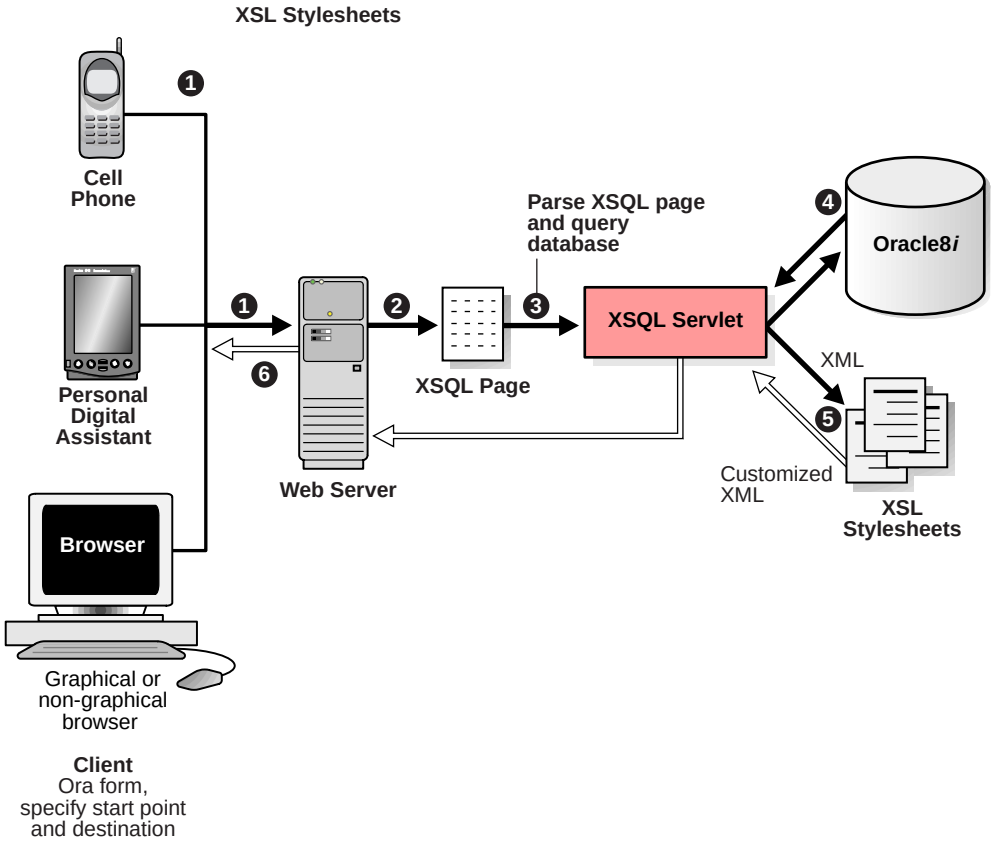
- SQL, the standard for accessing business data
- Oracle XSQL Servlet, which processes queries defined in XSQL pages. XSQL pages are XML documents that contain SQL code. XSQL Servlet outputs the result set as XML.
- XSLT, which defines an open standard for transforming XML for target devices.

This chapter describes how Flight Finder application was implemented. You can use these techniques in any Web-based application that:

- Receives requests from any client device on the Web.
- Delivers database content to multiple devices.
- Writes input from multiple devices back to the database.

Figure 8–1 shows how Flight Finder works.

Figure 8–1 XML Flight Finder



1. Using any supported client device, an end-user fills out a form to specify a starting point and a destination. The form's source code specifies an XSQL page to execute when the end-user submits the form.
2. The Web server invokes the XSQL Servlet with an XSQL Page.
3. The XSQL Servlet parses the XSQL page and queries the database.
4. The database returns the query results, which the XSQL Servlet converts to an XML document.
5. The XSQL Servlet transforms the XML by applying an XSL stylesheet appropriate for the end-user's client device.

6. The Web server returns the customized document to the client.

With Oracle8i, you can run Oracle XML components and applications built with them inside the database. For devices and applications that require a smaller database footprint, you can use Oracle8i Lite to store and retrieve XML data. You can also run these components on a middle tier such as Oracle Internet Application Server 8i, or on the client.

Flight Finder Queries the Database — Converts Results to XML

This section describes how Flight Finder queries the database and converts the result set to an XML document. Flight Finder application consists of XSQL Pages and XSL stylesheets:

- XSQL Pages define queries
- XSL stylesheets format the query results.

There is no Java code in the Flight Finder--it delegates processing chores to Oracle XSQL Servlet.

Flight Finder stores flight data in two tables, AIRPORTS and FLIGHTS.

- In AIRPORTS, the CODE column is the primary key.
- In FLIGHTS, the CODE column is the primary key, and the CODE_FROM and CODE_TO columns are foreign keys that reference AIRPORTS.CODE.

The following SQL code shows the structures of these tables (column names in bold are primary keys, column names in italics are foreign keys).

```
create table airports
(
 code varchar2(3),
 name varchar2(64)
 );

create table flights
(
code varchar2(6),
 code_from varchar2(3),
       code_to varchar2(3),
       schedule date,
       status varchar2(1),
        gate varchar2(2)
       );
```

Using XSQL Servlet to Process Queries and Output Result as XML

XSQL Servlet processes SQL queries and outputs the result set as XML.

It is implemented as a Java servlet and takes as input an XSQL page. This is an XML file containing embedded SQL queries. It uses XML Parser for Java and XML- SQL Utility for Java to perform many of its operations.

For example, the following code is from `fly.xsql`. It is XML with some special <xsql> tags for the XSQL Servlet to interpret.

`flightFinderResult` tag defines a structure that assigns values to parameters in a query. The tag also identifies a namespace for defining the xsql keyword and tells the XSQL servlet to use the (predefined) database connection named fly.

The code uses the `<xsql:query>` tag to define a query (the XSQL Servlet download includes a Help System that describes the syntax and options for each XSQL tag). The code uses two other parameters (FROM and TO) in the body of the query statement to store the names of cities chosen by the end-user.

Note: XSQL pages use the XSLT syntax {@param} to indicate a parameter.

Figure 8–2 shows the Flight Finder browser form and how it is used to enter FROM information (Los Angeles) and TO information (San Francisco).

Figure 8–2 Using XSQL Servlet to Process Queries and Output Result as XML: Entering FROM and TO on the Flight Finder Browser Form


```
<?xml version="1.0"?>
   ...
<flightFinderResult xmlns:xsql="urn:oracle-xsql" connection="fly"
lang="english">
     <xsql:set-stylesheet-param name="lang" value="{@lang}"/>
```

```
    <xsql:query tag-case="upper">
      <![CDATA[
      select F.code, F.code_from, A1.name as "depart_airport",
          F.code_to, To_char(F.schedule, 'HH24:MI') as "Sched",
          A2.name as "arrive_airport",
          Decode(F.Status, 'A', 'Available', 'B', 'Full', 'Available')
             as "Stat",F.Gate
             from flights F, airports A1, airports A2
             where to_number(To_Char(F.schedule, 'HH24MI')) >
                   to_number(To_Char(sysdate, 'HH24MI')) and
                   F.code_from = '{@FROM}' and F.code_to = '{@TO}' and
                   F.code_from = A1.code and F.code_to = A2.code
        ]]>
    ...
    </xsql:query>
   ...
</flightFinderResult>
```

The listing below shows a portion of the XML returned by the XSQL Servlet by processing the following URL. This is case-sensitive.

```
http://localhost:7070/fly.xsql?FROM=LAX&TO=SFO&xml-stylesheet=none
```

This URL tells the server to invoke the XSQL Servlet and process the file `fly.xsql` to find flights from LAX (Los Angeles) to SFO (San Francisco) without applying a stylesheet (a useful debugging technique because it shows the raw XML code, including error messages, if any, from the database).

The result is an XML document containing data from the rows in the result set (the following excerpt shows only the first row).

Tags ROWSET and ROW are defined by the XSQL Servlet. The tags for each row in a rowset (for example, CODE, CODE_FROM, and DEPART_AIRPORT) come from the names of columns in database tables.

```
<?xml version="1.0" ?>
       <flightFinderResult lang="english">
            <ROWSET>
               <ROW NUM="1">
                     <CODE>OA0307</CODE>
                     <CODE_FROM>LAX</CODE_FROM>
                     <DEPART_AIRPORT>Los Angeles</DEPART_AIRPORT>
                     <CODE_TO>SFO</CODE_TO>
                     <SCHED>12:04</SCHED>
                     <ARRIVE_AIRPORT>San Francisco</ARRIVE_AIRPORT>
                     <STAT>Available</STAT>
```

```
            <GATE>05</GATE>
          </ROW>
        ..
      </ROWSET>
    ...
  </flightFinderResult>
```

An XML document contains data and tags that describe the data, but no information about how to format the data for presentation. This may seem like a limitation at first glance, but it's actually a feature, and it's what makes XML so flexible. Once you have data in an XML document, you can format it any way you like.

Formatting XML with Stylesheets

Flight Finder applies an XSLT transformation to render the XML results in a format suitable for the end-user's client device. This section describes the process.

For general information about the relationships between XML, XSLT, and XSQL Servlet, see XSQL Pages and XSQL Servlet Release Notes on Oracle Technology Network (OTN), http://technet.oracle.com/tech/xml

One Stylesheet, One Target Device

Flight Finder uses XSL stylesheets to format the XML documents that represent query results. A stylesheet is itself an XML document that specifies how to process the nodes of another XML document. The processing instructions are defined in structures called templates, and a stylesheet formats a document by applying these templates to selected nodes.

For example, the XML document above contains nodes named ROWSET, ROW, CODE, etc. The following code (from flyHTMLdefault.xsl) shows how the stylesheet selects the CODE, DEPART_AIRPORT, and ARRIVE_AIRPORT nodes for each ROW in a ROWSET, and it applies templates to format the output.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version='1.0'>
...
<xsl:template match="/">
<html>
...
  <xsl:for-each select="flightFinderResult/ROWSET/ROW">
    <tr>
      <td><xsl:apply-templates select="CODE"/></td>
       <td><xsl:apply-templates select="DEPART_AIRPORT"/></td>
         <td><xsl:apply-templates select="ARRIVE_AIRPORT"/></td>
          ...
     </tr>
     </xsl:for-each>
     ...
  </html>
 </xsl:template>
<xsl:template match="CODE">Fly Oracle Airlines <xsl:value-of select="."/>
</xsl:template>
<xsl:template match="DEPART_AIRPORT">Leaving <xsl:value-of select="."/>
</xsl:template>
<xsl:template match="ARRIVE_AIRPORT">
 for <xsl:value-of select="."/>
```

```
</xsl:template>
...
</xsl:stylesheet>
```

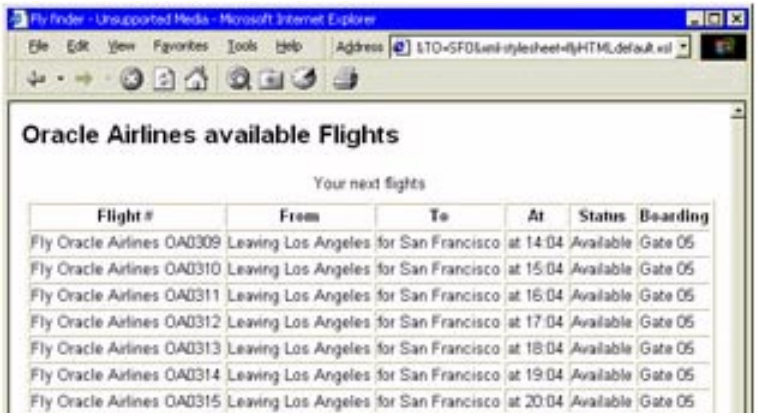
In this example, the formatting is simple: it just prepends a string to the contents of each node. For example, when the XSLT processor gets to the CODE node, it prepends the string "Fly Oracle Airlines " to the value of that node. The resulting HTML looks like this:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
...
<TR>
 <TD>Fly Oracle Airlines OA0309</TD>
 <TD>Leaving Los Angeles</TD>
 <TD>for San Francisco</TD>
 ...
</TR>
...
</HTML>
```

In a browser (enter the URL http://localhost:7070/fly/fly.xsql?FROM=LAX&TO=SFO&xml-stylesheet=flyHTMLdefault.xsl).

Figure 8–3 shows the results displayed on the browser after the stylesheet has been applied to the XML.

Figure 8–3 Flight Finder: Results After Formatting the XML with Stylesheets



Many Stylesheets, Many Target Devices

XSL stylesheets are the key to multiple devices, languages, and user interfaces. You can include multiple `<?xml-stylesheet?>` tags at the top of an XSQL Page, and each of those tags can define media and href attributes to associate a user agent with an XSL stylesheet (an HTTP request includes a user-agent header that identifies the device making the request). A processing instruction without a media attribute matches all user agents so it can be used as the fallback / default.

For example, the following XML code comes from fly.xsql. It includes several `<?xml-stylesheet?>` tags, including one that maps the stylesheet `flyVox.xsl` to the Motorola Voice Browser agent, and one that maps the `flyPP.xsl` stylesheet to the HandHTTP (Palm Pilot) agent.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" media="MSIE 5.0" href="flyHTML.xsl"?>
<?xml-stylesheet type="text/xsl" media="Motorola Voice Browser"
href="flyVox.xsl"?>
<?xml-stylesheet type="text/xsl" media="UP.Browser" href="flyWML.xsl"?>
<?xml-stylesheet type="text/xsl" media="HandHTTP" href="flyPP.xsl"?>
<?xml-stylesheet type="text/xsl" href="flyHTMLdefault.xsl"?>

<flightFinderResult xmlns:xsql="urn:oracle-xsql" connection="fly"
lang="english">
<xsql:stylesheet-param name="lang" value="{@lang}"/>
<xsql:query tag-case="upper">
...
```

```
</xsql:query>
...
</flightFinderResult>
```

The two listings below show the XSLT code to format one result set row each for a Palm Pilot (flyPP.xsl) and a voice browser device (flyVox.xsl).

XSLT Code From flyPP.xsl:

```
 ...
  <xsl:for-each select="flightFinderResult/ROWSET/ROW">
    <tr>
      <td>
        <a>
          <xsl:attribute name="href">
            #<xsl:value-of select="CODE"/>
          </xsl:attribute>
          <b><xsl:value-of select="CODE"/></b>
        </a>
      </td>
      <td><xsl:apply-templates select="SCHED"/></td>
      <td><xsl:apply-templates select="GATE"/></td>
    </tr>
  </xsl:for-each>
  ...
    <xsl:template match="CODE">
      <xsl:value-of select="."/>
    </xsl:template>
    <xsl:template match="SCHED">
      at <b><xsl:value-of select="."/></b>
    </xsl:template>
    <xsl:template match="GATE">
      gate <b><xsl:value-of select="."/></b>
    </xsl:template>
  ...
```

XSLT Code from flyVox.xsl:

```
...
<xsl:for-each select="flightFinderResult/ROWSET/ROW">
 <step><xsl:attribute name="name">
 step<xsl:value-of select="position()"/>
  </xsl:attribute>
   <prompt>
```

```
      <xsl:apply-templates select="CODE"/>
      <xsl:apply-templates select="SCHED"/>,
      <xsl:text>Do you take that one?</xsl:text>
    </prompt>
    <input type="OPTIONLIST" name="FLIGHT">
    <xsl:choose>
     <xsl:when test="position() = @NUM">
      <option>
       <xsl:attribute name="next">
        #<xsl:value-of select="CODE"/>
       </xsl:attribute>
       <xsl:text>Yes</xsl:text>
      </option>
       <xsl:if test="position() &lt; last()">
      <option>
       <xsl:attribute name="next">#step<xsl:value-of select="position() + 1"/>
       </xsl:attribute>
       <xsl:text>Next</xsl:text>
      </option>
     </xsl:if>
     <xsl:if test="position() &gt; 1">
      <option>
       <xsl:attribute name="next">#step<xsl:value-of select="position() - 1"/>
       </xsl:attribute>
       <xsl:text>Previous</xsl:text>
      </option>
     </xsl:if>
    </xsl:when>
   </xsl:choose>
  </input>
</step>
</xsl:for-each>
...
```

Localizing Output

When you invoke the Flight Finder through its portal (index.html), you can choose a language for prompts and labels.

The Flight Finder supports in English, French, Spanish, and German. To do this, it uses a parameter to identify the end-user's language of choice and passes it from HTML to XSQL to XSL, then it selects the appropriate text from a file of translated messages. For example, here is an overview of how the application tracks a user's language preference (French) and selects a label in that language:

1. `index.html` (The user clicks a link to choose a language):

   ```
   <a href="http://localhost:7070/xsql/fly/index.xsql?lang=french">Français</a>
   ```

2. `index.xsql` (The XSQL Page stores the user's choice in a parameter):

   ```
   <xsql:set-stylesheet-param name="lang" value="{@lang}"/>
   ```

3. `flyHTML.xsl` (The stylesheet uses the language choice parameter to select a message from the message file):

   ```
   <xsl:value-of select= "document('messages.xml')/messages/msg[@id=101 and
   @lang=$lang]"/>
   ```

4. `messages.xml` (The message file stores the translated messages):

   ```
   <msg id="101" lang="french">Prochains vols sur Oracle Airlines</msg>
   ```

The following listings show these steps in context.

index.html displays HREF links that invoke index.xsql with URLs for each supported language.

..

For Web-to-Go

```
<!-- Assumes default install to c:\xsql and Flight Finder files in c:\xsql\fly
-->
<ul>
 <li type="disc">
  <a href="http://localhost:7070/xsql/fly/index.xsql">English</a>
   </li>
   <li type="disc">
  <a
href="http://localhost:7070/xsql/fly/index.xsql?lang=french">Fran&ccedil;ais</a>
   </li>
   <li type="disc">
  <a
href="http://localhost:7070/xsql/fly/index.xsql?lang=spanish">Espa&ntilde;ol</a>
   </li>
   <li type="disc">
  <a href="http://localhost:7070/xsql/fly/index.xsql?lang=german">Deutsch</a>
   </li>
  </ul>
...
```

Next, the user's choice is extracted from the URL and plugged into a parameter in index.xsql. If the URL does not specify a language, a line in the following code sets it to English by default. This XSQL Page also defines a query (not shown here), which the XSQL Servlet sends to the database.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" media="Mozilla" href="indexHTML.xsl"?>
...
<index xmlns:xsql="urn:oracle-xsql" connection="fly" lang="english">
<xsql:set-stylesheet-param name="lang" value="{@lang}"/>
...
</index>
```

When the database returns the query results, the XSQL Servlet formats them by applying an XSLT transformation. The following code is from the stylesheet flyHTML.xsl. It includes a line that opens the message file (messages.xml) and selects message 101 for a specified language.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version='1.0'>
    <xsl:output media-type="text/html" method="html"/>
      <xsl:param name="lang" select="@lang"/>
       <xsl:template match="/">
          <html>
...
          <body>
          ...
<!-- Next available flights -->
<xsl:value-of select=
 "document('messages.xml')/messages/msg[@id=101 and @lang=$lang]"/>
  ...
         </body>
       </html>
     </xsl:template>
     ...
</xsl:stylesheet>
```

The XML code below comes from messages.xml. In this file, a message represents information (such as a label or a prompt) that the Flight Finder sends to the client. Messages are identified by ID numbers, and each message is translated into each supported language. The code below shows four translations of message 101. Notice that translations can include code for international character sets, as in the German version of the message. You may need to set your browser to display such

characters; for example, in Internet Explorer, choose View > Encoding > Western European (Windows).

```
<?xml version="1.0"?>
<messages>
...
  <msg id="101" lang="english">Oracle Airlines available flights</msg>
  <msg id="101" lang="french">Prochains vols sur Oracle Airlines</msg>
  <msg id="101" lang="spanish">Proximos vuelos sobre Oracle Airlines</msg>
 <msg id="101" lang="german">M&#246;gliche Fl&#252;ge mit Oracle Airlines</msg>
...
</messages>
```

XML to Database

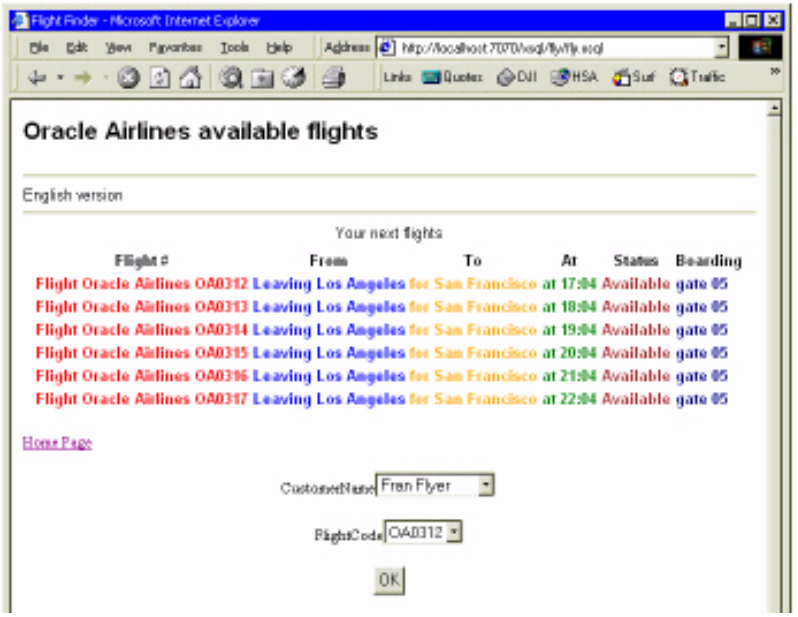
This section describes how the Flight Finder takes input from a user, converts it to XML, then writes it to the database.

1 Taking the User's Input

The first step is getting user input.

Figure 8–4 shows an HTML form that displays the results of a query about flights from Los Angeles to San Francisco, and provides drop-down lists of customer names and flight codes. The user chooses a name and a code, then clicks the OK button to book that flight for that customer, and the application writes the information to the database. This part of the application is only implemented for HTML and English.

Figure 8–4 Flight Finder: HTML Form Displaying Results of a Query About Flights From Los Angeles, to San Francisco



Here is the code from `fly.xsql` that populates drop-down lists named CustomerName and FlightCode with values from the database. The <form> tag

includes an action attribute that specifies bookres.xsql as the file to execute to process the values when the user submits the form.

The file `flyHTML.xsl` (not listed), provides the XSLT instructions for formatting the form as shown in the figure above.

...

```
<form action="bookres.xsql" method="post">
   <field name="CustomerName">
      <xsql:query rowset-element="dropDownList"
          row-element="listElem">
          <![CDATA[
             select unique name as "listItem"
               from customers
               order by name
               ]]>
            </xsql:query>
           </field>
            <field name="FlightCode">
             <xsql:query rowset-element="dropDownList"
                           row-element="listElem">
              <![CDATA[
               select F.code as "listItem",
               F.code as "itemId",
               A1.name as "depart_airport",
               A2.name as "arrive_airport"
                 from flights F,
                 airports A1,
                 airports A2
                 where to_number(To_Char(F.schedule, 'HH24MI')) >
                       to_number(To_Char(sysdate, 'HH24MI')) and
                       F.code_from = '{@FROM}' and
                       F.code_to = '{@TO}' and
                       F.code_from = A1.code and
                       F.code_to = A2.code
                 ]]>
              </xsql:query>
             </field>
             <sendRequest type="button" label="OK"/>
            </form>
        ...
```

2 Assign Values Acquired From User to Code Parameters

After getting values from the user, the next step is to assign those values to parameters in code. The following code comes from `bookres.xsql`.

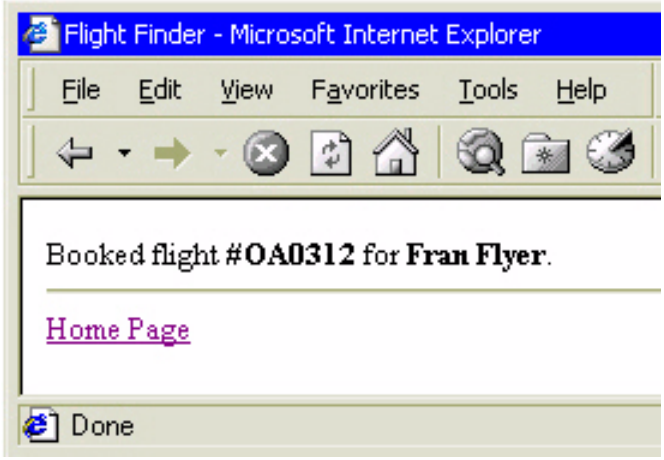
It stores the user's choices in parameters named CustomerName and FlightCode, and defines parameters named cust and code for passing the values to XSLT stylesheets. It also uses the `<xsql:dml>` tag to define a SQL statement that inserts a row into the CUSTOMERS table.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" media="Mozilla" href="bookresHTML.xsl"?>
<?xml-stylesheet type="text/xsl" media="MSIE 5.0" href="bookresHTML.xsl"?>
   <bookFlight xmlns:xsql="urn:oracle-xsql" connection="fly">
        <xsql:set-stylesheet-param name="cust" value="{@CustomerName}"/>
        <xsql:set-stylesheet-param name="code" value="{@FlightCode}"/>
        <xsql:dml>
        <![CDATA[
           insert into customers values
            ('{@CustomerName}', tripseq.NEXTVAL, '{@FlightCode}')
         ]]>
        </xsql:dml>
       ...
  </bookFlight>
```

3 Let User Know if Operation Succeeded

The last step is to let the user know whether the operation succeeded, in this case, whether the flight was booked as shown in.

Figure 8–5 Flight Finder: Notifying User that Flight Was Booked



The following code is from `bookresHTML.xsl`.

It declares parameters named cust and code to store values passed to it from bookres.xsql, then it uses those parameters to display a message to the user. The XSLT syntax for using such parameters is $param.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output media-type="text/html"/>
     <xsl:param name="cust"/>
       <xsl:param name="code"/>
        <xsl:template match="/">
          <html>
            <head>
            <title>Flight Finder</title>
            </head>
            <body>
              Booked flight #<b><xsl:value-of select="$code"/></b>
              for <b><xsl:value-of select='$cust'/></b>.
              <hr/>
             <xsl:apply-templates select="bookFlight/returnHome"/>
            </body>
           </html>
        </xsl:template>
       ...
</xsl:stylesheet>
```

Oracle Portal-to-Go

Instead of writing XSQL and XSL code yourself, you can use Oracle Portal-to-Go.

A component of the Oracle Internet Platform, Portal-to-Go provides everything you need to deliver Web content to any capable device. It transforms existing content to a device's native format, and it provides a portal interface for the end-user and can be developed on Oracle JDeveloper.

Portal-to-Go uses XML to isolate content acquisition from content delivery.

A Portal-to-Go portal includes the following components:

- Services that deliver data to mobile devices
- Adapters that convert HTML and RDBMS content to XML
- Transformers that convert XML to the appropriate markup language, including HTML, WML, TinyHTML, and voice mark-up language (VoxML).

For more information, including white papers, product documentation, and a free, downloadable version of the software, visit OTN's Portal-to-Go page.

> **See Also:** Chapter 7, "Personalizing Data Display With XML: Portal-to-Go".

Part IV

Data Exchange Using XML

Part IV begins with a description of Oracle Advanced Queuing and how its is used in B2B messaging applications. Several case studies are included to describe ways of implementing XML based data exchanges in B2B and B2C applications.

This part contains the following chapters:

9

Using Oracle Advanced Queuing (AQ) in XML Data Exchange

This chapter contains the following sections:

What is AQ?

Oracle Advanced Queuing (AQ) provides database integrated message queuing functionality. AQ :

- Enables and manages asynchronous communication of two or more applications using messages.
- Supports point-to-point and publish / subscribe communication models.

Integration of message queuing with Oracle8*i* database brings the integrity, reliability, recoverability, scalability, performance, and security features of Oracle8*i* to message queuing. Integration with Oracle8*i* also facilitates the extraction of intelligence from message flows.

How do AQ and XML Complement Each Other?

A key strength of XML is that you can use it as a common format for data exchange between applications. XML is self-describing and an application can share its data without any predefined knowledge of the applications receiving the data. XML is used to define messages communicated between two or more applications, while AQ is used to *enable and manage* that communication.

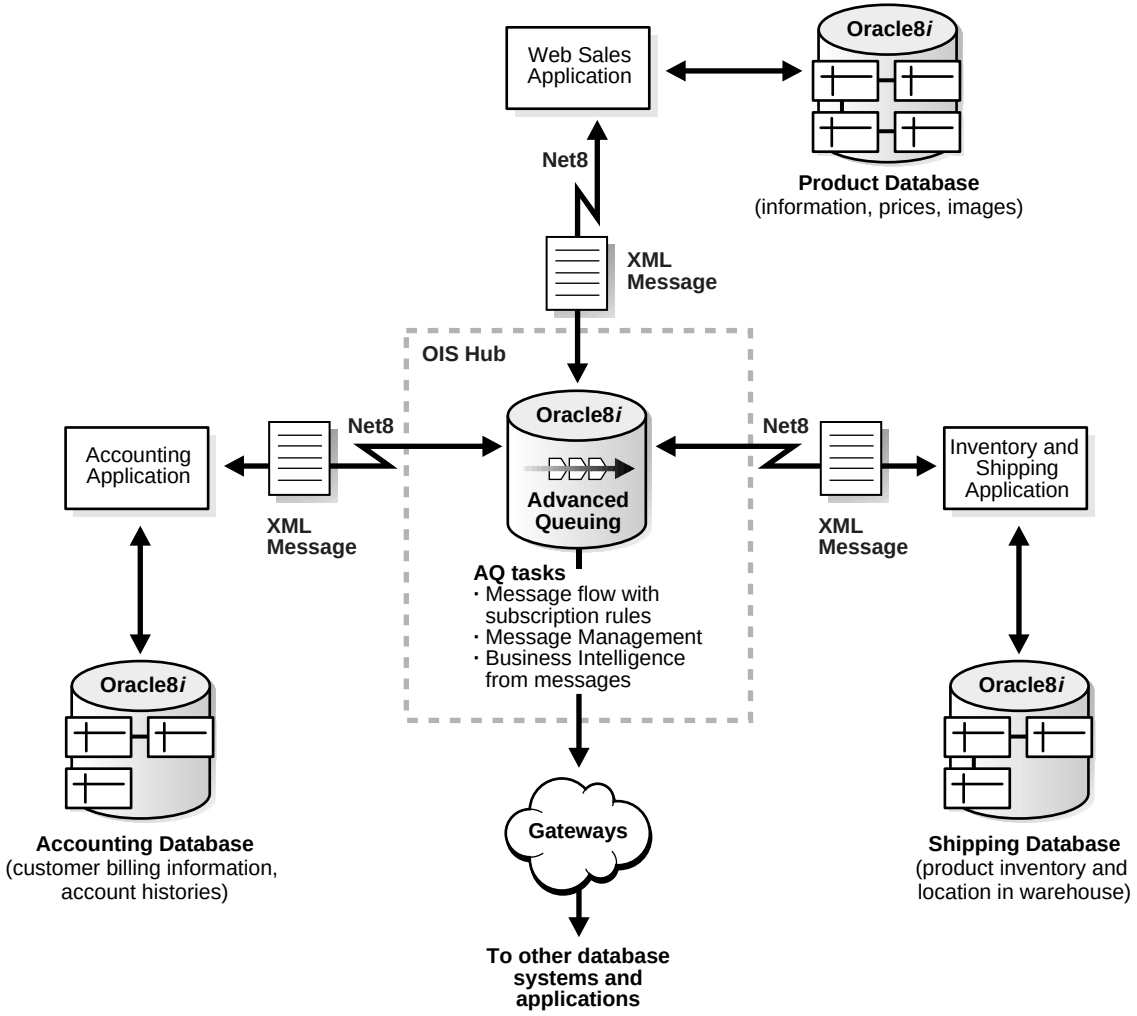
Figure 9–1 shows an Oracle8*i* database using AQ to communicate with three applications, with XML as the message payload. The main tasks performed by AQ in this scenario are:

- Message flow using subscription rules
- Message management
- Extracting business intelligence from messages

This is an *intra*-business scenario where XML messages are passed asynchronously among applications within an organization via AQ. Examples of this kind of scenario include sales order fulfillment and supply-chain management.

A similar scenario can apply to *inter*-business processes in which multiple integration hubs communicate over the Internet backplane. Examples of inter-business scenarios include travel reservations, coordination between manufacturers and suppliers, transferring of funds between banks, and insurance claims settlements, among others. Oracle uses this in its enterprise application integration products. XML messages are sent from applications to an AQ hub, here shown as an OIS hub. This serves as a "message server" for any application that wants the message. Through this hub and spoke architecture, XML messages can be communicated asynchronously to multiple loosely-coupled receiving applications.

Figure 9–1 Advanced Queueing and XML Message Payloads



AQ Enables Hub-and-Spoke Architecture for Application Integration

A critical challenge facing enterprises today is application integration. Application integration involves getting multiple departmental applications to cooperate, coordinate, and synchronize in order to execute complex business transactions.

Advanced Queuing enables hub-and-spoke architecture for application integration. It makes integrated solution easy to manage, easy to configure, and easy to modify with changing business needs.

Messages Can be Retained for Auditing, Tracking, and Mining

Message management provided by AQ is not only used to manage the flow of messages between different applications, but also, messages can be retained for future auditing and tracking, and extracting business intelligence.

Viewing Message Content With SQL Views

AQ also provides SQL views to look at the messages. These SQL views can be used to analyze the past, current, and future trends in the system.

Advantages of Using AQ

AQ provides the flexibility of *configuring communication* between different applications.

An XML message can be communicated using AQ in the following ways:

- Stored as an XML message in the AQ payload, as a VARCHAR or CLOB, or it can be compressed and stored as a BLOB.
- Using JMS, stored as a TextMessage or BytesMessage. It can also be processed into an object and stored as an ObjectMessage or an SQL ADT message.

AQ Example 1 (PL/SQL): XML Message as a CLOB in an AQ Message

This example shows you how to process an XML message stored in a CLOB in an AQ message. This example is in PL/SQL.

See Also: *Oracle8i Application Developer's Guide - Advanced Queuing*

Setting Up the AQ Environment

In order to set up the AQ environment, perform the following:

- Grant EXECUTE privileges on the PL/SQL packages DBMS_AQADM and DBMS_AQ
- Create the following tables:
  - Schema

- Queue tables
- Queues

AQ Example 1: Tasks Performed

This AQ example performs the following tasks:

1. Creates xml_payload_type to store XML messages in a CLOB
2. Creates AQ Queuetable to hold xml_payload_type messages
3. Creates AQ Queue
4. Adds Subscribers
5. Starts Queue for enqueue and Dequeue

Enqueuing a Message

1. Enqueues an empty CLOB
2. Selects CLOB locator using the enqueued message id
3. Populates the CLOB using DBMS_LOB package

Dequeuing a Message

1. Dequeues the message
2. Uses DBMS_LOB package to read the message

AQ Example 1: The PL/SQL Code

Here is the example:

```
CONNECT scott/tiger; -- User should have execute privileges on packages DBMS_
AQADM and DBMS_AQ.

-- Create an Oracle Object type to store an XML message

CREATE OR REPLACE TYPE xml_payload_type AS OBJECT (
-- if needed the message can also be parsed to extract some attributes to do
content-based routing on an AQ queue
xml_message CLOB;
);
```

```
-- CREATE an AQ Queuetable with XML payload
BEGIN
dbms_aqadm.create_queue_table(
queue_table => 'xmlmsg_queuetable',
        sort_list =>'priority,enq_time',
        comment => 'demonstrate XML message in an AQ queue',
        multiple_consumers => TRUE,
        queue_payload_type => 'xml_payload_type',
        compatible => '8.1');
END;
/

-- CREATE an AQ Queue for XML messages
BEGIN
dbms_aqadm.create_queue (
   queue_name              => 'xmlmsg_queue',
   queue_table             => 'xmlmsg_queuetable');
END;
/

-- Add an subscriber to XML message queue
BEGIN
dbms_aqadm.add_subscriber(
queue_name => 'xmlmsg_queue',
subscriber => 'xml_subscriber');
END;
/

-- Start queue for enqueuing and dequeuing
BEGIN
dbms_aqadm.start_queue('xmlmsg_queue');
END;
/

-- Enqueue an XML message
DECLARE
enqopt dbms_aq.enqueue_options_t;
msgprop dbms_aq.message_properties_t;
enq_msgidRAW(40);
sales_order     xml_payload_type;
order_loc       CLOB;
XML_msg         VARCHAR2(200);
msgsize Number;
BEGIN
sales_order := xml_payload_type (empty_clob());
```

```
dbms_aq.enqueue(
   queue_name            => 'xmlmsg_queue', -- IN
   enqueue_options       => enqopt,  -- IN
   message_properties    => msgprop, -- IN
   payload               => sales_order, -- IN
   msgid                 => enq_msgid); -- OUT
SELECT t.user_data.xml_message INTO order_loc
   FROM xmlmsg_queuetable t
   where t.msgid = enq_msgid;

-- Put an XML message in the lob
XML_msg := '<xml> </xml>';
dbms_lob.write(order_loc, 12, 1, XML_msg);

OMMIT;
END;
/


-- Dequeue an XML message
declare
dequeue_options     dbms_aq.dequeue_options_t;
message_properties  dbms_aq.message_properties_t;
message_handle      RAW(16);
message             xml_payload_type;
buffer varchar2(100);
msglen number;
begin
   dequeue_options.consumer_name := 'xml_subscriber';
   dequeue_options.wait          := dbms_aq.NO_WAIT;
   dbms_aq.dequeue(
                    queue_name           => 'xmlmsg_queue',
                    dequeue_options      => dequeue_options,
                    message_properties   => message_properties,
                    payload              => message,
                    msgid                => message_handle);
   commit;
           msglen := dbms_lob.getlength(message.xml_message);
   dbms_lob.read(message.xml_message, msglen, 1,
buffer);
dbms_output.put_line(buffer);
end;
/
```

AQ Example 2 (Java): Processing an XML Message Using JMS (Publish - Subscribe)

XML messages can be enqueued and dequeued from an AQ queue using JMS (Java Messaging Standard). Example 2 shows you how to publish and subscribe an XML message as a JMS TextMessage. Refer to *Oracle8i Application Developer's Guide - Advanced Queuing* for more examples.

AQ Example 2: Processing an XML Message Using JMS —Tasks Performed

Setting Up the AQ JMS Environment

See *Oracle8i Application Developer's Guide - Advanced Queuing Chapter 13, "*JMS Administrative Interface- Basic Operations" for information on setting up the AQ JMS environment.

Publish (Enqueue) an XML Message using JMS

The following lists the tasks performed by Example 2 using JMS to publish an XML message:

1. Gets the name and location of Topic on a certain subject
2. Gets ConnectionFactory for JMS provider that hosts desired topic
3. Opens a Connection to the JMS provider using ConnectionFactory
4. Creates a JMS Session
5. Gets a Topic object using the JMS Session
6. Creates the Message to be published
7. Creates a TopicPublisher to send messages
8. Publishes the Message to the Topic

Receive (Dequeue) an XML Message using JMS

The following lists the tasks performed by Example 2 using JMS to receive (subscribe) an XML message:

1. Gets the name and location of Topic on certain subject
2. Gets a ConnectionFactory of JMS provider that hosts desired Topic
3. Opens a Connection to the JMS provider using ConnectionFactory

4. Creates a JMS Session
5. Gets a Topic object using the JMS Session
6. Creates a TopicSubscriber to receive desired message
7. Waits for messages - using blocking receive call or by registering MessageListener

AQ Example 2: Processing an XML Message Using JMS — Java Code

Here is the Java code listing using JMS to process an XML message.

```
public void publishXMLMessage (String host, String ora_sid, int port, String
driver) throws JMS Exception
{
TopicConnectionFactory tc_fact;
TopicConnection        t_conn;
TopicSession           jms_sess;
Topic                  mlmsg_topic;
TextMessage            xmlmsg;
TopicPublisher         xmlmsg_publisher;

// create a ConnectionFactory
tc_fact = AQjmsFactory.createTopicConnectionFactory(
                    host, ora_sid, port, driver);
//create a Topic Connection
t_conn = tc_fact.createTopicConnection ("scott", "tiger");
//create a JMS Session
jms_sess = t_conn.createTopicSession(true, 0);

// Get a Topic object
xmlmsg_topic = ((AQjmsSession)jms_sess).getTopic("scott", "xmlmsg_topic");
//create Topic publisher
xmlmsg_publisher = jms_sess.createPublisher(null);
// create XML text message
xmlmsg =jms_sess.createTextMessage();
//Publish message
xmlmsg_publisher.publish(xmlmsg_topic, xmlmsg);
jms_sess.commit();
}

public void receiveXMLMessage (String host, String ora_sid, int port, String
driver) throws JMS Exception
{
TopicConnectionFactorytc_fact;
```

```
TopicConnectiont_conn;
TopicSessionjms_sess;
Topicxmlmsg_topic;
TextMessagexmlmsg;
TopicSubscriberxmlmsg_subscriber;

// create a ConnectionFactory
tc_fact = AQjmsFactory.createTopicConnectionFactory(
                    host, ora_sid, port, driver);
//create a Topic Connection
t_conn = tc_fact.createTopicConnection ("scott", "tiger");
//create a JMS Session
jms_sess = t_conn.createTopicSession(true, 0);

// Get a Topic object
xmlmsg_topic = ((AQjmsSession)jms_sess).getTopic("scott", "xmlmsg_topic");
//create Topic subscriber
xmlmsg_subscriber = jms_sess.createDurableSubscriber(xmlmsg_topic, "XmlMsg_
Subscriber");
// create XML text message
xmlmsg =(TextMessage)xmlmsg_subscriber.receive();
//Process Text Message

jms_sess.commit();
}
```

Frequently Asked Questions (FAQs): XML and Advanced Queuing

Multiple Format Messages: Create an Object Type and Store as Single Message

Question

We are exchanging XML documents from one business area to another using Oracle Advanced Queuing. Each message received or sent includes an XML header, XML attachment (XML data stream), DTDs, and PDF files. We need to store all this information, including some imagery files, in the database table, in this case, the queuetable.

Can we enqueue this message into an Oracle queue table as one record or one piece? Or do we have to enqueue this message as multiple records, such as one record for XML data streams as CLOB type, one record for PDF files as RAW type, ...

Then somehow specify these set of records are correlated? Also we want to ensure that we dequeue this.

Answer

You can achieve this in the following ways:

- You can either define an object type with (CLOB, RAW,...) attributes, and store it as a single message
- You can use the AQ message grouping feature and store it in multiple messages. But the message properties will be associated with a group. To use message grouping feature, all messages must be the same payload type.

Question 2

Does this mean that we specify the payload type as CLOB first, then enqueue and store all the pieces, XML message data stream, DTDs, and PDF,... as a single message payload in the Queue table? If so, how can we separate this single message into individual pieces when we dequeue this message?

Answer 2

No. You create an object type, for example:

```
CREATE TYPE mypayload_type as OBJECT (xmlDataStream CLOB, dtd CLOB, pdf BLOB);
```

Then store it as a single message.

Adding New Recipients After Messages are Enqueued

Question

We want to use the queue table to support message assignments. For example, when other business areas send messages to Oracle, they do not know who should be assigned to process these messages, but they know the messages are for Human Resources (HR). So all messages will go to the HR supervisor.

At this point, the message has been enqueued in the queue table. The HR supervisor is the only recipient of this message, and the entire HR staff have been pre-defined as subscribers for this queue). Can the HR supervisor add new recipients, namely additional staff, to the message_properties.recipient_list on the existing the message in the queue table?

We do not have multiple consumers (recipients) when the messages are enqueued, but we want to replace the old recipient, or add new recipients after the message has already been in the queue table. This new message will then be dequeued by the new recipient. Is this workable? Or do we have to remove the message from old recipient, then enqueue the same message contents to the new recipient?

Answer

You cannot change the recipient list after the message is enqueued. If you do not specify a recipient list then subscribers can subscribe to the queue and dequeue the message.

In your case, the new recipient should be a subscriber to the queue. Otherwise, you will have to dequeue the message and enqueue it again with the new recipient.

XML and AQ

Question

In the OTN document, "Using XML in Oracle Database Applications, Part 4, Exchanging Business Data Among Applications" Nov. 1999, it says that an Oracle database can enqueue and dequeue XML messages and process them. How does it do this?

Do I have to use XML-SQL Utility (XSU) in order to insert an XML file into a table before process it, or can I enqueue an XML file directly, parse it, and dispatch its

messages via the AQ process? Must I use XML-SQL Utility every time I want to INSERT or UPDATE XML data into an Oracle Database?

Answer

AQ supports enqueing and dequeing objects. These objects can have an attribute of type CLOB containing an XML Document, as well as other interested "factored out" metadata attributes that might make sense to send along with the message. Refer to the latest AQ document, *Oracle8i Application Developer's Guide - Advanced Queuing,* to get specific details and see more examples.

Retrieving and Parsing JMS Clients with XML Content From AQ

Question

We need a tool to parse messages, JMS clients with XML content, from an AQ queue and then update tables and fields in an ODS (Operational Data Store). In short, we want to retrieve and parse XML documents and map specific fields to database tables and columns.

Is *inter*media Text a solution?

Answer

The easiest way to do this is using Oracle XML Parser for Java and Java Stored Procedures in tandem with AQ inside Oracle8*i*.

Question 2

We can use XML-SQL Utility if we go with a custom solution. Our main concentration is supply-chain. We want to get metadata information such as, AQ enqueue / dequeue times, JMS header information,.... based on queries on certain XML tag values. Can we just store the XML in a CLOB and issue queries using *inter*media Text?

Answer 2

Your question said "parsing" messages, so that threw me off. It also mentioned putting message metadata in regular tables.

- If you store XML as CLOBs then you can definitely search it using *inter*Media Text, but this only helps you find a particular message that matches a criteria.

- If you need to do aggregation operations over the metadata, view the metadata from existing relational tools, or use normal SQL predicates on the metadata, then having it "only" stored as XML in a CLOB is not going to be good enough.

You can combine *inter*Media Text XML searching with some amount of redundant metadata storage as "factored out" columns and use SQL statements that combine normal SQL predicates with the *inter*Media Text CONTAINS() clause to have the best of both.

Ensuring that an Enqueue is Successful?

Question

I am looking for ways to ensure the enqueue is successful. I am deploying AQ on Oracle8*i*. After successfully executing the enqueue statement in a PL/SQL procedure, I found that the queue had not actually started. Also, there was no error message returned after the enqueue. I have somehow lost the message.

Is there a way to deal with this situation?

Answer

Try the following:

1. After creating the queue, issue the dbms_aqadm.start_queue command.
2. After enqueuing a message, issue a COMMIT.

10

B2B: How *i*Procurement Uses XML to Offer Multiple Catalog Products to Users

This chapter contains the following sections:

Introduction to Oracle Internet Procurement (*i*Procurement)

Oracle *i*Procurement is a web-based catalog content exchange application that helps users find and order available products and services. It helps automate the entire purchasing life-cycle from sourcing to procurement to payment. *i*Procurement also has a web-shopping interface with Express and Power checkout options and a built-in Internet Procurement Connector for transactions with third party ERP systems.

*i*Procurement supports several catalog content management methods. Users can choose any combination of the following:

- Buyer-hosted content
- Supplier- hosted content
- Third party hosted marketplace

All of these content-management methods use XML for their web-based data (content) exchange technology. This chapter describes each of these content-management methods and associated XML documents. It also provides several examples.

Various Suppliers Load Their Catalogs into the Unified Catalog Tables

*i*Procurement uses XML to load catalogs received from various suppliers or catalog service providers, into "unified catalog tables" in the *i*Procurement database. See Figure 10–1.

XML also transfers security or authentication objects when communicating with external catalog sourcing companies.

- *Problem*: Build a catalog that seamlessly offers multiple supplier catalogs of goods and services to users via the web.
- *Business Solution*: *i*Procurement with Oracle Applications including, Oracle Exchange, Oracle XML technology, PL/SQL programs, XML interface.
- *Oracle XML Components Used*: XML Parser for Java

Oracle Internet Procurement Solution

Oracle's Internet Procurement *solution* leverages the combined power of Oracle Internet Procurement, Oracle Supplier Network, and Oracle Exchange. Companies can buy all types of goods and services at the best value,

make better strategic decisions, and improve the bottom line. This solution helps automate and centralize procurement functions such as sourcing, approval routing, and payments while decentralizing the requisitioning and receiving processes.

Oracle Exchange is anopen business-to-business online marketplace that allows companies to buy and sell goods and services using any purchasing method, with content and related services provided by the Oracle Supplier Network. Oracle Exchange is accessible by any company, regardless of size or industry, and does not require any Oracle software.

More Information About Internet Procurement and Related Products

You can find more information about *i*Procurement at the following sites:

http://www.oracle.com/products/

http://www.oracle.com/applications/internetprocurement/index.html

You can find more information about Oracle Exchange at:

http://www.oracle.com/applications/exchange/index.html

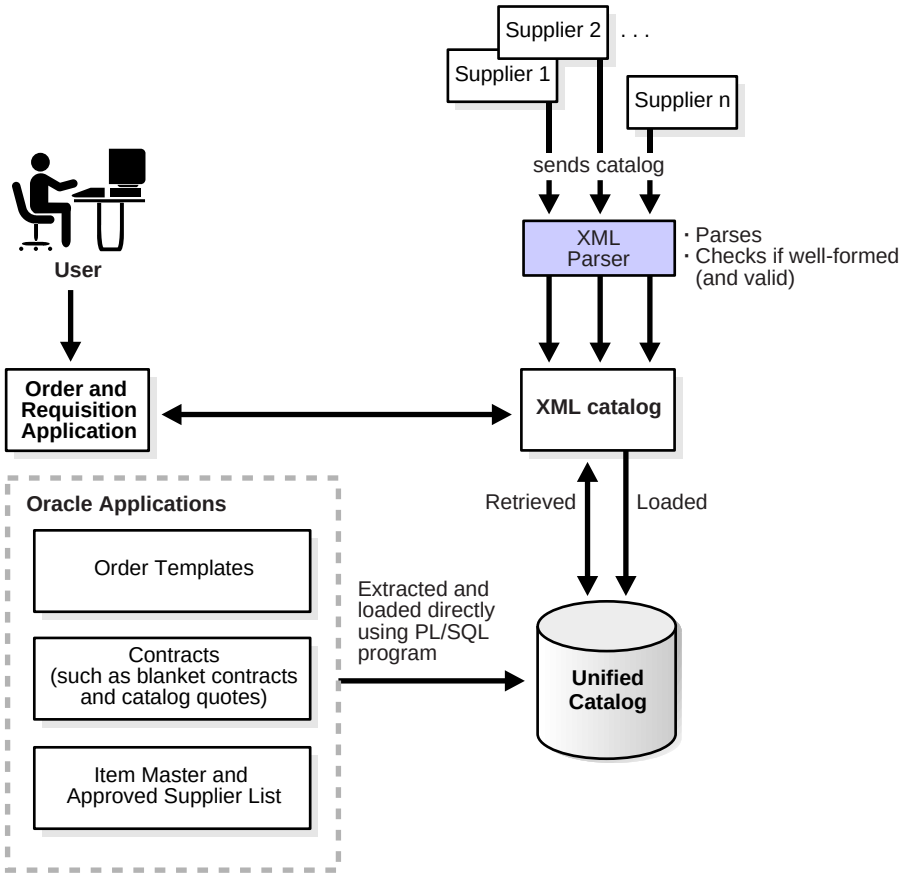
Buyer-Hosted Content

With buyer-hosted content management, you can request items by searching for products or services in a single "unified catalog" without having to first select a supplier. Content in the unified catalog is received from either of the following sources:

- Loaded directly from Oracle Applications, including the following:
  - Item Master
  - Approved Supplier List
  - Requisition Template
  - Contract information
- From any third party content provider via the XML interface described in this chapter. One Oracle partner, for example, uses the XML interface to publish private, high-quality electronic supplier catalogs.

The use of XML in the buyer-hosted catalog exchange process is illustrated in Figure 10–1.

Figure 10–1 Buyer-Hosted Content Management



Oracle Applications data is extracted and loaded directly into the unified catalog by a PL/SQL concurrent program. Hence in this case, it is not necessary to generate and parse the XML document.

Supplier-Hosted Catalogs and Marketplaces

While creating a requisition, users can also navigate to and select items from external third party catalogs. These can be hosted by a supplier or catalog provider. In this case, users select a link for the desired catalog, select items and then return to

*i*Procurement where they can request additional items, make changes, or complete their order.

You can use Oracle XML interface, which encompasses secure user authentication and item selection to link any externally hosted catalog to *i*Procurement. This XML interface is used to provide access to a secure, private catalog that aggregates high-quality content from multiple suppliers.

Figure 10–2 Supplier-Hosted Catalog XML Exchange

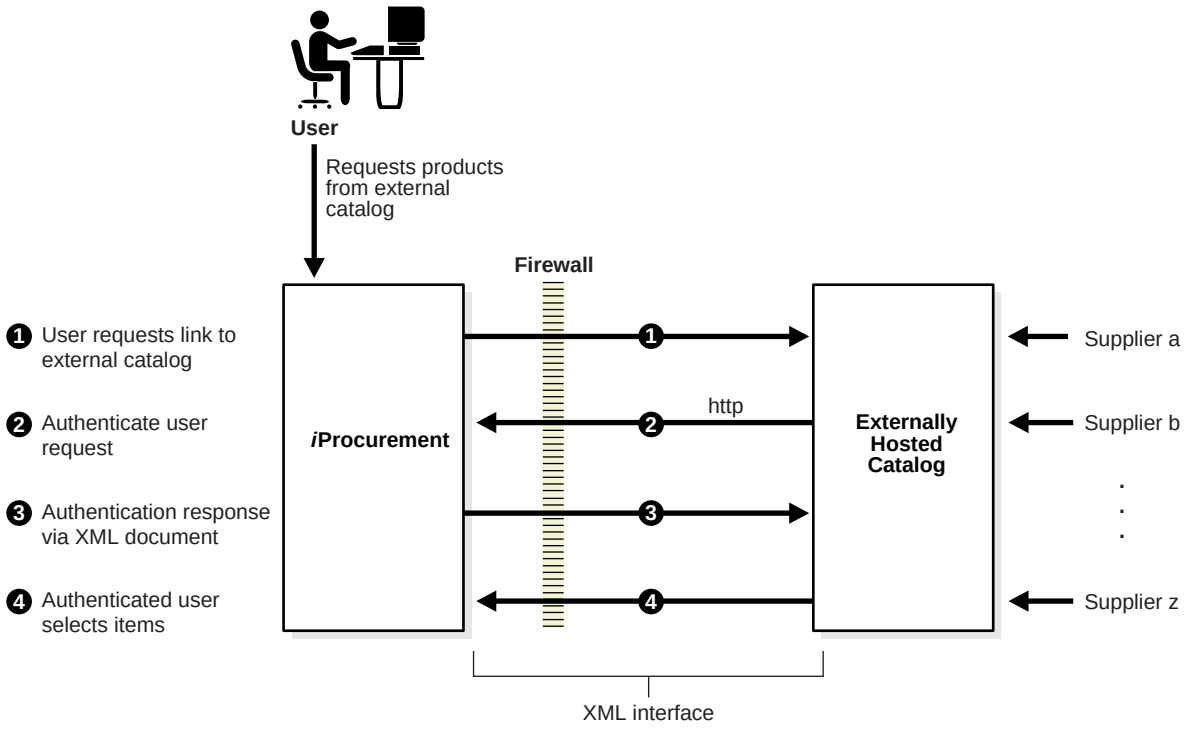


Figure 10–2 illustrates the data exchange process when users select items from supplier-hosted on-line catalogs. While shopping in *i*Procurement, the following exchange protocol occurs:

1. Requestor (user) selects a link and navigates to an external catalog. At this time, encrypted session information and an authentication URL are provided.

2. The external catalog asks *i*Procurement, via a secure HTTP call, to authenticate the user.
3. *i*Procurement responds to the authentication request with an XML document indicating whether the user was successfully authenticated. If so, additional information about the requestor is sent.
4. The authenticated user shops at the external site, selects items, and finally indicates that they are ready to add their items to the requisition. This triggers the external catalog to transmit the selected items to *i*Procurement in an XML document that is parsed and converted to requisition lines. The user returns to *i*Procurement to complete their order.

*i*Procurement: XML Parser for Java

*i*Procurement includes Oracle's XML Parser for Java (installed on the middle tier but separate from the Internet Application Server (*i*AS). This parser checks if an XML document is well-formed, and optionally, if it is valid.

Internationalization

*i*Procurement is fully NLS compliant, however, the product does not support multiple languages in releases 10.7 and 11. For this reason, XML document data must be specified in the base language of the installation.

> **Note:** To avoid character set conversion data loss between the XML document and the database, specify a character set encoding that is both supported by the Oracle XML Parser and compatible with the database installation character set.

Language Identification

Both XML documents support language specification using the `xml:lang` attribute as described in *the Extensible Markup Language (XML) 1.0* W3C recommendation. Refer to http://www.w3.org/TR for all published, draft, or proposed recommendations.

The following table, extracted from the XML 1.0 specification, describes in general terms how language is identified:

```
LanguageID:=Langcode (- Subcode)*
Langcode:=ISO639Code | IanaCode | UserCode
ISO639Code:=([a-z] | [A-Z]) ([a-z] | [A-Z])
```

```
IanaCode:= (‘i’ | ‘I’) ‘-‘ ([a-z] | [A-Z]) +
UserCode:= (‘x’ | ‘X’) ‘-‘ ([a-z] | [A-Z]) +
SubCode:=([a-z] | [A-Z]) +
```

According to the XML 1.0 specification, Langcode can be any of the following:

- A two-letter language code as defined by ISO 639, “Codes for the representation of the names of languages”
- A language identifier registered with the Internet Assigned Numbers Authority (IANA)
- A language identifier agreed on between parties in private use

There may be any number of Subcode segments, however, if the first subcode exists and consists of two letters then it must be a country code from ISO 3166, “Codes for the representation of names of countries.”

*i*Procurement Language Identification

To convert from the XML document language specification to Oracle Applications language codes, specify the following:

- An ISO 639 language code
- An ISO 3166 country code

The following example shows how to set the language to English and the country to the United States:

```
<e xml:lang=”EN-US”>
```

The `xml:lang` declaration applies all attributes and content of the element where it is specified, unless overridden with another instance of `xml:lang` on a specific attribute. Since releases 10.7 through 11i of *i*Procurement expect a single language, this should be specified for the root element in each document type.

*i*Procurement Character Set Encoding

The XML Parser processor is informed of the character set used in an XML document via the encoding parameter of the XML declaration:

```
<?xml . . . encoding=’UTF-8’ . . . ?>
```

If no document encoding is specified in the XML declaration, UTF-8 is assumed.

Any other ASCII or EBCIDIC based encodings supported by the JDK can be used, however, they must be specified in the format required by JDK instead of as official character set names defined by IANA.

See http://OTN.oracle.com/tech/xml for the latest XML Parser supported character sets. Current support character sets and other Parser specifications also listed in Appendix C, "XDK for Java: Specifications and Cheat Sheets".

Buyer-Hosted Catalogs

This section describes the XML specification for loading the *i*Procurement unified catalog from external sources. The specification supports the following functionality:

- Catalog schema (item classification data) modifcation
  - Add, rename, or delete a category
  - Add, rename, or delete a descriptor (a category attribute)
- Item addition and category assignment
- Item deletion and modification

Document Type Definition (DTD)

Special characters and non-XML markup must be escaped for the XML parser to function correctly. Specifically, the & and <> characters must be escaped with a CDATA tag.

```
<![CDATA[ your data here ]]> inserted in any tag for special characters.

<SCHEMA>
   <CATEGORY ACTION="DELETE">
       <NAME><![CDATA[Pen & Pencil Gifts Sets]]></NAME>
   </CATEGORY>
</SCHEMA>
```

Also, if there are any " (double quote) characters within the data itself, these should be replaced by the following character sequence: `&quot;`

For example:

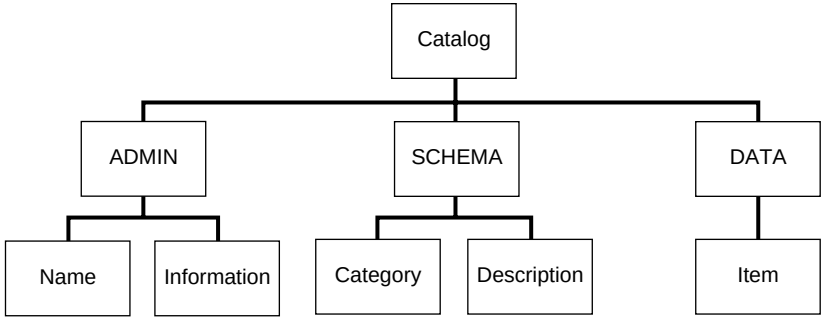
```
<itemDescription><![CDATA[6&quot; diameter pipe]]></itemDescription>
```

Catalog information is divided into three main categories:

- DTD Admininistrative Information: <ADMIN> on page 10-11
- DTD Schema Information: <SCHEMA> on page 10-11
- DTD: Item Information on page 10-15

Figure 10–3 shows the DTD hierarchical diagram of the catalog:

Figure 10–3 Buyer-Hosted Catalog: DTD Hierarchical Diagram



*i*Procurement Example 1: DTD for Buyer-Hosted Catalog

Here is the Buyer-Hosted catalog DTD:

```
<?xml version="1.0"?>
<!DOCTYPE CATALOG [

<!ELEMENT CATALOG ( ADMIN, SCHEMA?, DATA? ) >
<!ATTLIST CATALOG xml:lang NMTOKEN #IMPLIED >

<!ELEMENT ADMIN ( NAME, INFORMATION) >
<!ELEMENT SCHEMA (CATEGORY | The  DESCRIPTOR)* >
<!ELEMENT DATA (ITEM)*>

<!ELEMENT NAME (#PCDATA) >
<!ELEMENT INFORMATION ( DATE, SOURCE )  >
<!ELEMENT DATE (#PCDATA) >
<!ELEMENT SOURCE (#PCDATA) >

<!ELEMENT CATEGORY (NAME | KEY | TYPE | UPDATE  )* >
<!ATTLIST CATEGORY ACTION (ADD|DELETE|UPDATE) #REQUIRED>
<!ELEMENT DESCRIPTOR (NAME | KEY | UPDATE | OWNER  | TYPE )* >
<!ATTLIST DESCRIPTOR ACTION (ADD|DELETE|UPDATE) #REQUIRED>
<!ELEMENT OWNER (NAME?, KEY? ) >
<!ELEMENT KEY (#PCDATA) >
<!ELEMENT TYPE (#PCDATA) >

<!ELEMENT ITEM (OWNER?, NAMEVALUE*, UPDATE ) >
<!ATTLIST ITEM ACTION (ADD | DELETE | UPDATE) #REQUIRED>
```

```
<!ELEMENT UPDATE (NAME | KEY | NAMEVALUE )* >

<!ELEMENT NAMEVALUE ( NAME, VALUE ) >
<!ELEMENT VALUE (#PCDATA)* >
]>
```

Note: *i*Procurement does *not* validate the DTD during the parsing process and therefore does not require a copy of the DTD in the code tree. It is provided here for reference only.

DTD Admininistrative Information: <ADMIN>

The <ADMIN> section is used to identify the catalog. This section is required.

```
<ADMIN>
   <NAME>Business Essential Reference Catalog</NAME>
   <INFORMATION>
 <DATE>![CDATA[02-FEB-99]]</DATE>
<SOURCE>eContent Manager</SOURCE>
   </INFORMATION>
</ADMIN>
```

Information contained in this section is source catalog-specific and does not impact the loading of data for other catalogs. The <ADMIN> fields are described in Table 10–1.

Table 10–1 DTD: <ADMIN> Fields

Field	Required	COMMENT
Name	Required	This is the name of the catalog
Date	Required	This is the catalog creation or modification date. Note: This should match the default database format for the Applications instance.
Source	Required	This is the company, person, or tool that authored the XML document.

DTD Schema Information: <SCHEMA>

In the catalog, the schema consists of categories and descriptors.

- Categories: Categories and the items they contain also have associated descriptive attributes. A category can be thought of as an item classification, such as pens, paper, books, and software.
- Descriptors: Every catalog has two kinds of descriptors: "base" and "local."
  - Base descriptors apply universally to all items in the catalog regardless of category, such as, Item Description, SKU, Price, and so on.
  - Local descriptors are relevant only for items belonging to a specific category, such as, "CPU Speed" for the category "Computer", or "Ink Color" for the category "Pens".

A category or a descriptor can consist of a name, a key, or both. When a key is provided, it is given precedence in identifying the category or descriptor. the DTD's <SCHEMA> caegory and descriptor fields are described in Table 10–2.

Table 10–2 DTD: <SCHEMA> Category and Descriptor Fields

Field	Required	COMMENT
Name	Conditionally required (one of the 2 must be provided)	Descriptive name for the category or descriptor
Key		Unique identifier for the category or descriptior
Owner	Required for descriptors; optional for categories	For descriptors, this is the category (name or key) with which you want to associate the descriptor. Setting the owner value to "Root" or 0 designates the descriptor as being applicable for all items/categories. This is known as a "base" or "root" descriptor. For categories, defining the Owner creates a hierarchical relationship that can be expressed in the Table of Contents.

Type	Optional	If used for a Descriptor, this field indicates that data type of the descriptor value. Valid options include: ■ STRING (Default) ■ NUMBER ■ INTERNATIONAL (indicates that this descriptor value will ultimately have translations when MLS is implemented)

The catalog schema can be modified by the following sequence. A <SCHEMA> tag has an attribute that specifies a command and sub-tags that define a key and actionable data. The commands are:

- ADD
- UPDATE
- DELETE

The examples below are enclosed in the <SCHEMA> tag.

*i*Procurement Example 2: DTD <SCHEMA> — Adding a Category and Descriptor to the Category

The following example adds a new category and descriptors to the new category.

```
<SCHEMA>
   <CATEGORY ACTION="ADD">
       <NAME>Pen Gift Sets</NAME>
       <KEY>PEN_GIFT_SETS</KEY>
   </CATEGORY>
   <DESCRIPTOR ACTION="ADD">
       <OWNER>
         <NAME>Pen Gift Sets</NAME>
       </OWNER>
         <NAME>Package Type</NAME>
       <TYPE>String</TYPE>
   </DESCRIPTOR>
</SCHEMA>
```

*i*Procurement Example 3: DTD <SCHEMA>: Deleting a Category or Descriptor

Categories and descriptors require a NAME, or KEY, tag to identify what is to be deleted. For descriptors, this identity is required for the category that contains the descriptor.

```
<SCHEMA>
   <CATEGORY ACTION="DELETE">
      <NAME>Pen Gifts Sets</NAME>
   </CATEGORY>
   <CATEGORY ACTION="DELETE">
      <NAME>Pen Gift Collections</NAME>
      <KEY>3245</KEY>
   </CATEGORY>

   <CATEGORY ACTION="DELETE">
       <KEY>Laptop Computer</KEY>
   </CATEGORY>

   <DESCRIPTOR ACTION="DELETE">
       <OWNER>
       <NAME>Pens</NAME>
       </OWNER>
       <NAME>Barrel Color</NAME>
   </DESCRIPTOR>

   <DESCRIPTOR ACTION="DELETE">
       <OWNER>
       <KEY>22343</KEY>
       </OWNER>
       <NAME>Barrel Color</NAME>
       <KEY>887665</KEY>
   </DESCRIPTOR>
</SCHEMA>
```

*i*Procurement Example 4: DTD <SCHEMA> — Updating Category or Descriptor

Categories and descriptors require a name, or key, tag to identify what is to be updated. For descriptors, the name, or key, tag is required for the category that contains the descriptor. Here is an example:

```
<SCHEMA>
   <CATEGORY ACTION="UPDATE" >
     <NAME>Pen Gifts Set</NAME>
     <UPDATE>
```

```
      <NAME>Pen Gift Sets</NAME>
    </UPDATE>
  </CATEGORY>

  <CATEGORY ACTION="UPDATE">
    <KEY>C440911</KEY>
    <UPDATE>
    <NAME>Compressor Motors</NAME>
    </UPDATE>
  </CATEGORY>
</SCHEMA>
```

DTD: Item Information

Table 10–3 lists the DTD's ITEM information fields.

Table 10–3 DTD ITEM> Information

Field	Required	COMMENT
Owner	Optional if the action is Delete; Required for other actions	Associates an item with a specific category.
Name / Value - Name	Required	Associates an item with a descriptor. The following base descriptor names are seeded by Oracle *i*Procurement and can be used when adding items: ■ Description ■ UOM ■ Sup Part Num ■ Mfg Part Num ■ Sup Name ■ Mfg Name ■ Price / / What about Long Description and Picture? Not seeded by Oracle but certainly handled specially by Requisite in the user interface

Name / Value - Value	Required	Associated value for the specified descriptor name.

Data items can be added to a category, deleted, and modified. All data items must be placed between the <DATA></DATA> tag. The identity feature (specifying a <NAME> and / or <KEY>) allows the wholesale change of values by effectively wildcarding updates. The update example below provides a method to change all items with a manufacturer Bic to Bic, Inc.

*i*Procurement Example 5: DTD ITEM — Adding Items Using <ITEM ACTION"ADD">

This is an example of adding items to the *i*Procurement DTD using <ITEM ACTION="ADD"> marker.

```
<DATA>
   <ITEM ACTION="ADD">
<OWNER>
<NAME>Pens</NAME>
</OWNER>
<NAMEVALUE>
<NAME>Mfg Name</NAME>
<VALUE>Bic</VALUE>
</NAMEVALUE>
<NAMEVALUE>
<NAME>Barrel Color</NAME>
<VALUE>Blue</VALUE>
</NAMEVALUE>
   </ITEM>

   <ITEM ACTION="ADD">
<OWNER>
<NAME>Pencils</NAME>
</OWNER>
<NAMEVALUE>
<NAME>Mfg Name</NAME>
<VALUE>Bic</VALUE>
</NAMEVALUE>
   </ITEM>
</DATA>
```

*i*Procurement Example 6: DTD ITEM — Deleting Items Using <ITEM ACTION="DELETE">

This is an example of deleting items from the *i*Procurement DTD using <ITEM ACTION="DELETE"> marker.

```
<DATA>
   <ITEM ACTION="DELETE">
<OWNER>
<NAME>Pens</NAME>
</OWNER>
<NAMEVALUE>
<NAME>Mfg Name</NAME>
<VALUE>Bic</VALUE>
</NAMEVALUE>   <!- -removes all items by Bic mfg from Pens - ->
<NAMEVALUE>
<NAME>Barrel Color</NAME>
<VALUE>Blue</VALUE>
</NAMEVALUE>
   </ITEM>
</DATA>
```

*i*Procurement Example 7: DTD ITEM — Updating Items Using <ITEM ACTION="UPDATE")

This is an example of updating items from the *i*Procurement DTD using <ITEM ACTION="UPDATE"> marker.

When updating and item, the owner tag is optional. This allows for updates of multiple items with a specific value.

```
<DATA>
   <-- Updating Item by Item -->
   <ITEM ACTION="UPDATE">
   <OWNER>
   <NAME>Pens</NAME>
   </OWNER>
    <NAMEVALUE>
      <NAME>Mfg Name</NAME>
      <VALUE>Bic </VALUE>
    </NAMEVALUE>
    <NAMEVALUE>
      <NAME>Barrel Color</NAME>
      <VALUE>Blue</VALUE>
    </NAMEVALUE>
```

```
    <UPDATE>
    <NAMEVALUE>
      <NAME>Mfg Name</NAME>
      <VALUE>Bic Inc.</VALUE>
    </NAMEVALUE>
    <NAMEVALUE>
      <NAME>Barrel Color</NAME>
      <VALUE>Red</VALUE>
    </NAMEVALUE>
    </UPDATE>
   </ITEM>
   <--   This section below updates items with Mfg Name= Bic -->
   <ITEM ACTION="UPDATE">
    <NAMEVALUE>
      <NAME>Mfg Name</NAME>
      <VALUE>Bic </VALUE>
    </NAMEVALUE>
   </ITEM>
</DATA>
```

Supplier Hosted Catalogs and Marketplaces

This section describes the specifications (data elements and file format) for any external catalog to transmit selected item information to Oracle. It describes the following, and is divided into 4 parts:

- Data Element Definition
- Order Line XML Definition
- "HTML Specification" and format for transmitting the XML data
- User Authentication

Data Element Definition

This section describes the data elements for the requisition order line file that is transmitted from the 3rd party catalog to Oracle once the user has completed item selection in the remote catalog. The data elements are grouped into 6 logical segments:

- **Contract** - identifies contract information for the selected item
- **Item** - identifies the item, order quantity and other descriptive item information
- **Category** - identifies the item's category code and catalog source (catalog provider or host)
- **Price** - identifies unit price information (in catalog currency or the user's functional currency) for the item
- **Supplier** - identifies the item's supplier
- **Additional Attributes** - lets you specify additional, custom information about the item (this maps to the Descriptive Flexfield columns in the Requisition Lines table)

Definitions

All data elements are either optional, required or conditional.

Table 10–4 Data Elements are Optional, Required, or Conditional

Data Element type	Description

Required	Required fields must be provided by the 3rd party. This data should be sufficient to complete the transaction.
Optional	Optional fields will improve the accuracy of the transaction, however, the system will derive, cross-reference, calculate or defaul the values
Conditional	This status indicates that under certain conditions, the data must be provided. These conditions are specified in the 'Required' column
Type	Possible values are: ■ Number (real or integer) ■ Character (the string length is defined for every field. If the submitted string is longer than the specified length, it will be truncated) ■ Date (all dates should be in the format YYYYMMDD
Cross-Reference	Indicates if a cross-reference to the Oracle EDI Gateway is necessary to obtain a recognized value.

Standard Codes

Table 10–5 lists the standard codes referenced in the Order Line.

Table 10–5 Standard Codes Referenced in the Order Line

Code	Description
D-U-N-S ® Number	Dun & Bradstreet has developed a nine-digit identification sequence commonly used as a company identifier in EDI and global electronic commerce transactions. A large organization is likely to have many different D&B D-U-N-S numbers since each business location may have its own unique identifier. For more information on Dun & Bradstreet, Inc. visit http://www.dnb.com.
UN/SPSC	The United Nations Development Programme and Dun & Bradstreet's have combined to define a standardized, open system for classifying products and services: UN/Standard Product & Service Code (UN/SPSC). The system uses a hierarchical structure with 5 levels and approximately 8,000 total classifications. For more information on the UN/SPSC codes, visit http://www.unspsc.org

Contract Data Elements

Table 10–6 Contract Data Elements

Field	Required	Type	Comment
Supplier Contract Number	Conditiontal: at least one of these contract numbers must	Character (25)	Contract number as defined in the supplier's system.
Buyer Contract Number	be specified	Character (20)	Contract number as defined in the buyer's system.
Buyer Contract Line Number	Optional	Number	Contract line item number in the buyer's system This can be derived from the ASL sourcing rules.
Catalog Type	Optional	Character (25)	Specify one of the following values: 'CONTRACTED' 'NONCONTRACTED'

Item Data Elements

Table 10–7 Item Data Elements

Field	Required	Type	Comment

Line Type	Optional	Character (25)	Specify one of the following values: 'GOODS' - specify the price, quantity and unit of measure. This is the default value 'SERVICES QUANTITY' - for rate-based services. Similar to goods; specify the price per unit of service and the quantity 'SERVICES AMOUNT' - for amount-based services. In Oracle Purchasing, the Price is set to 1 and the actual amount is entered in the Quantity field
Supplier Item Number	Conditional: at least one of the 3 item numbers must	Character (25)	
Manufacturer Item Number	be specified (Supplier, Manufacture r or Buyer)	Character (25)	Manufacturer Name is required if Manufacturer Item Number is specified
Buyer Item Number		Character (25)	Item number as defined in the buyer's system
Buyer Item Revision	Optional	Character (3)	Item revision as defined in the buyer's system
Item Description	Required	Character (240)	
Quantity	Optional	Number	Defaults to 1
Buyer Unit of Measure	Conditional: at least one of these 2 fields must be provided	Character (25)	Unit of measure code or description as defined in the buyer's system. Cross reference: EDI Gateway
Supplier Unit of Measure		Character (25)	Unit of measure code or description as defined in the supplier's system. Cross reference: EDI Gateway
Supplier Unit of Measure Quantity	Optional	Number	A quantity associated with the supplier's unit of measure. Cross reference: EDI Gateway

Manufacturer Name	Conditional	Character (40)	Required if Manufacturer Item Number is specified.
Hazard Class	Optional	Character (40)	UN and Department of Transportation provides standards. Cross reference: EDI Gateway

Category Data Elements

Table 10–8 Category Data Elements

Field	Required	Type	Comment
SPSC Category Code	Conditional: at least of 1 of these 3 values is required	Character (30)	Category code as defined in the UN / SPSC standard Cross reference: EDI Gateway
Supplier Category Code		Character (30)	Category code in the supplier system.
Buyer Category Code		Character (50)	Category code in the buyer system.
Catalog Source	Required	Character (30)	Name or code for the catalog provider or host (which could be different from the item's supplier). This value is labeled "Catalog Trading Partner" in the XML Schema code

Price Data Elements

Table 10–9 Price Data Elements

Field	Required	Type	Comment
Currency	Optional	Character (30)	Defaults to the functional (user's) currency as defined in the buyer's system. The codes are as defined in ISO. Cross reference: EDI Gateway

Unit Price	Required	Number	The price should match the current effective contract if one exists
Rate	Optional	Number	Conversion rate from catalog currency to the user's functional currency. If the specified currency does not = the functional currency, the rate will be determined as follows: If the item is contracted and a fixed conversion rate is specified in the contract, this rate is used The supplier can provide the rate information If this field is left blank and no fixed rate is specified on the contract, the rate can be obtained using internal referencing schemes (e.g. daily rates)
Rate Date	Conditional: Required only if rate is provided	Date	Use the format YYYYMMDD
Rate Type	Conditional: Required only if Rate is provided		Cross reference: EDI Gateway

Supplier Data Elements

Table 10–10 Supplier Data Elements

Field	Required	Type	Comment
Supplier DUNS	Conditional: at least 1 of these 4 values must be specified	Character (30)	Supplier number as defined by Dun & Bradstreet. Cross reference: EDI Gateway
Supplier Name		Character (80)	Supplier name
Supplier Number		Number	Supplier number as defined in the buyer's system

Supplier Trading Partner Code		Character (30)	Custom code assigned to a supplier to set them up as a trading partner
Supplier Site	Conditional: Required if Supplier DUNS is not provided	Character (15)	
Contact Name	Optional	Character (80)	Contact at the supplier site who can address questions about this transaction
Contact Phone	Optional	Character (20)	Contact phone number

Additional Attributes

Table 10–11 Additional Attributes

Field	Required	Type	Comment
Custom Attribute1	Optional	Character (150)	Definition determined by buyer and seller
Custom Attribute2	Optional	Character (150)	Definition determined by buyer and seller
Custom Attribute3	Optional	Character (150)	Definition determined by buyer and seller
Custom Attribute4	Optional	Character (150)	Definition determined by buyer and seller
Custom Attribute5	Optional	Character (150)	Definition determined by buyer and seller
Custom Attribute6	Optional	Character (150)	Definition determined by buyer and seller
Custom Attribute7	Optional	Character (150)	Definition determined by buyer and seller
Custom Attribute8	Optional	Character (150)	Definition determined by buyer and seller
Custom Attribute9	Optional	Character (150)	Definition determined by buyer and seller
Custom Attribute10	Optional	Character (150)	Definition determined by buyer and seller
Custom Attribute11	Optional	Character (150)	Definition determined by buyer and seller

Custom Attribute12	Optional	Character (150)	Definition determined by buyer and seller
Custom Attribute13	Optional	Character (150)	Definition determined by buyer and seller
Custom Attribute14	Optional	Character (150)	Definition determined by buyer and seller
Custom Attribute15	Optional	Character (150)	Definition determined by buyer and seller

Order Line XML Definition

Use the XML schema below as a template for formatting the data elements discussed in the previous section. The XML file will be passed through the Oracle XML parser which will extract the data and create requisition lines in *i*Procurement.

Please note that, although not explicitly stated in the schema, all attribute values should be enclosed in single quotes. For example, the schema statement:

```
<attribute name="categoryCodeIdentifier" atttype="ENUMERATION" values "SPSC
SUPPLIER BUYER"/>
```

should result in the following XML statement:

```
<category categoryCodeIdentifier='SPSC'>
```

Enclose All Data in CDATA Tags

All data must be enclosed in CDATA tags. Because data can contain special characters - single quotes, double quotes, and so on, this is necessary to ensure data integrity for and during parsing. For example, the schema statement:

```
<elementType id="manufacturerName">
     <string/>
     <description>the name of the manufacturer</description>
</elementType>
```

should result in the following XML statement:

```
<manufacturerName><![CDATA[Bob's Factory]]></manufacturerName>
```

*i*Procurement Example 8: Order Line XML Schema

```
<?xml version='1.0'?>
<s:schema id='OrderLinesDataItems'>
<elementType id="catalogTradingPartner">
     <string/>
     <description>Unique trading partner code in requisition
system</description>
</elementType>

<elementType id="contractNumber">
     <string/>
     <description>contract in which the item exists</description>
</elementType>
<elementType id="buyerContractLineNum">
```

```
     <string/>
     <description> line number of the item on the buyer contract </description>
</elementType>
<elementType id="catalogType">
     <string/>
     <description>catalog type: CONTRACTED/NONCONTRACTED</description>
</elementType>
<elementType id="supplierContract">
     <elementType ="#contractNumber"/>
     <description>supplier contract identifier for the line item </description>
</elementType>
<elementType id="buyerContract">
     <elementType ="#contractNumber"/>
     <description>buyer contract identifier for the line item </description>
</elementType>
<elementType id="contract">
     <attribute name=" contractNumberIdentifier" atttype="ENUMERATION" values=
"KNOWN UNKNOWN INFORMATIONAL NONE"/>
     <group groupOrder="OR">
          <elementType ="#supplierContract"/>
          <elementType ="#buyerContract"/>
          <elementType ="#buyerContractLineNumber"/>
          <elementType ="#catalogType" occurs "OPTIONAL "/>
     </group>
     <description>contract information for the line item </description>
</elementType>

<elementType id="itemID">
     <string/>
     <description>the item number in the chosen catalog/system</description>
</elementType>
<elementType id=" supplierItemNumber">
     <elementType ="#itemID"/>
     <description>supplier item number information</description>
</elementType>
<elementType id="manufacturerName">
     <string/>
     <description>the name of the manufacturer</description>
</elementType>
<elementType id=" manufacturerItemNumber">
     <elementType ="#itemID"/>
     <elementType ="#manufacturerName"/>
     <description>manufacturer item number information</description>
</elementType>
<elementType id="buyerItemRevision">
```

```
     <string/>
     <description> the buyer's item revision code(optional)</description>
</elementType>
<elementType id="buyerItemNumber">
     <elementType ="#itemID"/>
     <elementType ="#buyerItemRevision"/>
     <description>buyer item number information</description>
</elementType>
<elementType id="itemNumber">
     <group groupOrder="OR">
          <elementType ="#supplierItemNumber"/>
          <elementType ="#manufacturerItemNumber"/>
          <elementType ="#buyerItemNumber"/>
     </group>
     <description>the item number in the chosen catalog/system</description>
</elementType>
<elementType id="itemDescription">
     <string/>
     <description>the description of the item</description>
</elementType>
<elementType id="quantity">
     <number/>
     <description> quantity of the item (optional)</description>
</elementType>
<elementType id="buyerUnitOfMeasure">
     <string/>
     <description> unit of measure of the item on buyer system</description>
</elementType>
<elementType id="supplierUOMType">
     <string/>
     <description> unit of measure of the item on supplier system</description>
</elementType>
<elementType id="supplierUOMQuantity">
     <number/>
     <description> quantity associated with supplier unit of measure
(optional)</description>
</elementType>
<elementType id="supplierUnitOfMeasure">
     <elementType ="#supplierUOMType"/>
     <elementType ="#supplierUOMQuantity" occurs "OPTIONAL "/>
     <description> item information on supplier system equivalent to buyer unit
of measure</description>
</elementType>
<elementType id="UnitOfMeasure">
     <group groupOrder="OR">
```

```
          <elementType =″#buyerUnitOfMeasure″/>
          <elementType =″#supplierUnitOfMeasure″/>
</group>
     <description> unit of measure of the item</description>
</elementType>
<elementType id=″hazardClass″>
     <string/>
     <description> the hazard class ID  (optional)</description>
</elementType>
<elementType id=″item″>
     <attribute name =″lineType″ atttype=″ENUMERATION″ values= “GOODS
AMOUNTBASEDSERVICES RATEBASEDSERVICES “ default=″GOODS″>
     <elementType =″#itemNumber″/>
     <elementType =″#itemDescription″/>
     <elementType =″#quantity″ occurs “OPTIONAL “>
          <default>1</default>
     <elementType>
     <elementType =″#unitOfMeasure″ />
     <elementType =″#hazardClass″ occurs “OPTIONAL “/>
     <description>identifies the item</description>
</elementType>

<elementType id=″categoryCode″>
     <string/>
     <description>the code for the category</description>
</elementType>
<elementType id=″category″>
    <attribute name=″categoryCodeIdentifier″ atttype=″ENUMERATION″ values “SPSC
SUPPLIER BUYER″/>
     <elementType =″#categoryCode″/>
     <description>indicates item source catalog & category code</description>
</elementType>

<elementType id=″currency″>
     <string/>
     <description>ISO currency code</description>
</elementType>
<elementType id=″unitPrice″>
     <number/>
     <description>the price per unit of measure</description>
</elementType>
<elementType id=″rateDate″>
     < date/>
     <description> date of rate shown</description>
</elementType>
```

```
<elementType id="rateType">
     < String/>
     <description> type of rate</description>
</elementType>
<elementType id="rate">
     <attribute name="rateDefinition" atttype="ENUMERATION" values "FUNCTIONAL
CONTRACT SUPPLIER BUYER"/>
     < number/>
     <elementType ="#rateDate"/>
     <elementType ="#rateType"/>
     <description> conversion rate between currencies</description>
</elementType>
<elementType id="price">
     <elementType ="#currency"/>
     <elementType ="#unitPrice"/>
     <elementType ="#rate" occurs "OPTIONAL "/>
     <description>item unit price in appropriate currency</description>
</elementType>

<elementType id="supplierSite">
     <string/>
     <description>the supplier's site</description>
</elementType>
<elementType id="supplierDUNS">
     <string/>
     <description>the supplier's DUNS number</description>
</elementType>
<elementType id="supplierName">
     <string/>
     <elementType ="#supplierSite"/>
     <description>the supplier's name</description>
</elementType>
<elementType id="supplierNumber">
     <number/>
     <elementType ="#supplierSite"/>
     <description>supplier number as described in buyer system</description>
</elementType>
<elementType id="supplierTradingPartner">
     <string/>
     <description>supplier trading partner code</description>
</elementType>
<elementType id="contactName">
     <string/>
     <description> contact person at supplier site</description>
</elementType>
```

```
<elementType id="contactPhone">
     <string/>
     <description>phone number of contact</description>
</elementType>
<elementType id="supplier">
     <group groupOrder="OR">
          <elementType ="#supplierDUNS"/>
          <elementType ="#supplierName"/>
          <elementType ="#supplierNumber"/>
          <elementType ="#supplierTradingPartner"/>
     </group>
     <elementType ="#contactName" occurs "OPTIONAL "/>
     <elementType ="#contactPhone" occurs "OPTIONAL "/>
     <description>identifies suppliers</description>
</elementType>

<elementType id="languageCode">
     <string/>
     <description>the language code</description>
</elementType>
<elementType id="language">
     <elementType ="#languageCode" occurs "OPTIONAL "/>
     <description>language used to enter information for this item</description>
</elementType>

<elementType id="attribute1">
     <string/>
     <description>optional extra line attribute</description>
</elementType>
<elementType id="attribute2">
     <string/>
     <description>optional extra line attribute</description>
</elementType>
<elementType id="attribute3">
     <string/>
     <description>optional extra line attribute</description>
</elementType>
<elementType id="attribute4">
     <string/>
     <description>optional extra line attribute</description>
</elementType>
<elementType id="attribute5">
     <string/>
     <description>optional extra line attribute</description>
</elementType>
```

```
<elementType id="attribute6">
     <string/>
     <description>optional extra line attribute</description>
</elementType>
<elementType id="attribute7">
     <string/>
     <description>optional extra line attribute</description>
</elementType>
<elementType id="attribute8">
     <string/>
     <description>optional extra line attribute</description>
</elementType>
<elementType id="attribute9">
     <string/>
     <description>optional extra line attribute</description>
</elementType>
<elementType id="attribute10">
     <string/>
     <description>optional extra line attribute</description>
</elementType>
<elementType id="attribute11">
     <string/>
     <description>optional extra line attribute</description>
</elementType>
<elementType id="attribute12">
     <string/>
     <description>optional extra line attribute</description>
</elementType>
<elementType id="attribute13">
     <string/>
     <description>optional extra line attribute</description>
</elementType>
<elementType id="attribute14">
     <string/>
     <description>optional extra line attribute</description>
</elementType>
<elementType id="attribute15">
     <string/>
     <description>optional extra line attribute</description>
</elementType>
<elementType id="additionalAttributes">
     <elementType ="#attribute1" occurs "OPTIONAL "/>
     <elementType ="#attribute2" occurs "OPTIONAL "/>
     <elementType ="#attribute3" occurs "OPTIONAL "/>
     <elementType ="#attribute4" occurs "OPTIONAL "/>
```

```
     <elementType =″#attribute5″ occurs “OPTIONAL “/>
     <elementType =″#attribute6″ occurs “OPTIONAL “/>
     <elementType =″#attribute7″ occurs “OPTIONAL “/>
     <elementType =″#attribute8″ occurs “OPTIONAL “/>
     <elementType =″#attribute9″ occurs “OPTIONAL “/>
     <elementType =″#attribute10″ occurs “OPTIONAL “/>
     <elementType =″#attribute11″ occurs “OPTIONAL “/>
     <elementType =″#attribute12″ occurs “OPTIONAL “/>
     <elementType =″#attribute13″ occurs “OPTIONAL “/>
     <elementType =″#attribute14″ occurs “OPTIONAL “/>
     <elementType =″#attribute15″ occurs “OPTIONAL “/>
     <description>additional information about the item</description>
</elementType>

<elementType id=″orderLine″>
     <elementType =″#contract″/>
     <elementType =″#item″/>
     <elementType =″#category″/>
     <elementType =″#price″/>
     <elementType =″#supplier″/>
     <elementType =″#language″/>
     <elementType =″#additionalAttributes″/>
     <description>Order line sent to requisition server</description>
</elementType>

<elementType id=″OrderLinesDataElements″>
     <elementType =″#catalogTradingPartner″/>
     <elementType =″#orderLine″ occurs “ZEROORMORE″/>
     <description>complete order line sent to requisition server</description>
</elementType>
</s:schema>
```

Note: In this example, the ‘catalogTradingPartner’ in the XML header maps to the ‘catalog source’ field for each requisition line.

For an explanation of all the terms used in the schema, see the XML Data Spec: http://www.w3.org/TR/1998/NOTE-XML-data-0105/

Oracle tailored several definitions, for example enumeration. Enumeration is used as a case statement switch in some elements of the above schema. This decides what data the element will contain. For example, in “contract” the enumerated attribute value can be either KNOWN, UNKNOWN, INFORMATIONAL or NONE. It is these values that decide what the element will contain as values.

- If it is KNOWN or INFORMATIONAL: It can contain either the buyer contract information, the supplier contract information or both.
- If it is UNKNOWN or NONE: There are simply no fields.

*i*Procurement Example 9: XML — One Order Line for the Full Schema Specification

This is an example of one order line covering the full schema specification including optional tags. It has approximately 50 tags and is 2.5K bytes.

```
<?xml version='1.0'?>
<OrderLinesDataElements xml:lang='EN-US'>
<catalogTradingPartner><![CDATA[ABCCatalogServices]]></catalogTradingPartner>
<orderLine>
<contract contractNumberIdentifier='KNOWN'>
    <supplierContract>
        <contractNumber><![CDATA[12323634634]]></contractNumber>
    </supplierContract>
    <buyerContract>
        <contractNumber><![CDATA[987654321]]></contractNumber>
    </buyerContract>
    <buyerContractLineNumber><![CDATA[99]]></buyerContractLineNumber>
    <catalogType><![CDATA[CONTRACTED]]></catalogType>
</contract>

<item lineType='GOODS'>
    <itemNumber>
       <supplierItemNumber>
            <itemID><![CDATA[B1324]]></itemID>
       </supplierItemNumber>
       <manufacturerItemNumber>
            <itemID><![CDATA[X456]]></itemID>
            <manufacturerName><![CDATA[Bob's Factory]]></manufacturerName>
        </manufacturerItemNumber>
        <buyerItemNumber>
            <itemID><![CDATA[2222XY]]></itemID>
            <buyerItemRevision><![CDATA[4]]></buyerItemRevision>
        </buyerItemNumber>
    </itemNumber>
    <itemDescription><![CDATA[Purple and Red]]></itemDescription>
    <quantity><![CDATA[999]]></quantity>
    <unitOfMeasure>
        <buyerUnitOfMeasure><![CDATA[ea]]></buyerUnitOfMeasure>
        <supplierUnitOfMeasure>
           <supplierUOMType ><![CDATA[each]]></supplierUOMType>
```

```
            <supplierUOMQuantity><![CDATA[1]]></supplierUOMQuantity>
        </supplierUnitOfMeasure>
    </unitOfMeasure>
    <hazardClass><![CDATA[2768]]></hazardClass>
</item>

<category categoryCodeIdentifier='SPSC'>
    <categoryCode><![CDATA[5149-9908-00]]></categoryCode>
</category>

<price>
    <currency><![CDATA[USD]]></currency>
    <unitPrice><![CDATA[50.99]]></unitPrice>
    <rate rateDefinition='CONTRACT'>
        <rateDate><![CDATA[19981210]]></rateDate>
        <rateType><![CDATA[corporate]]></rateType>
    </rate>
</price>

<supplier>
    <supplierName><![CDATA[ACME Hot Air Balloons]]></supplierName>
    <supplierSite><![CDATA[Kinshasa]]></supplierSite>
    <supplierTradingPartner><![CDATA[Traders R Us</supplierTradingPartner>
    <contactName><![CDATA[Ramanujam Kondetimmanahalli]]></contactName>
    <contactPhone><![CDATA[3015061111]]></contactPhone>
</supplier>

<language>
    <languageCode><![CDATA[AmEnglish]]></languageCode>
</language>

<additionalAttributes>
    <attribute1><![CDATA[additional information 1 ]]></attribute1>
    <attribute2><![CDATA[additional information 2 ]]></attribute2>
    <attribute3><![CDATA[additional information 3 ]]></attribute3>
    <attribute4><![CDATA[additional information 4 ]]></attribute4>
    <attribute5><![CDATA[additional information 5 ]]></attribute5>
    <attribute6><![CDATA[additional information 6 ]]></attribute6>
    <attribute7><![CDATA[additional information 7 ]]></attribute7>
    <attribute8><![CDATA[additional information 8 ]]></attribute8>
    <attribute9><![CDATA[additional information 9 ]]></attribute9>
    <attribute10><![CDATA[additional information 10]]></attribute10>
    <attribute11><![CDATA[additional information 11]]></attribute11>
    <attribute12><![CDATA[additional information 12]]></attribute12>
    <attribute13><![CDATA[additional information 13]]></attribute13>
```

```
    <attribute14><![CDATA[additional information 14]]></attribute14>
    <attribute15><![CDATA[additional information 15]]></attribute15>
</additionalAttributes>

</orderLine>
</OrderLinesDataElements>
```

*i*Procurement Example 10: XML — Two-Item Transaction Example

This is an example of a typical two-item transaction with the following items:

- First item is a "Hard drive - 540MB IDE" with a price of 485.99, quantity of 34, supplier name of A-1 Lighting.
- Second item is a "High speed assembly machine" with a price of 12575.99, quantity of 9, supplier name of JCN Technologies.

"ABC Catalog Services" is the external third party catalog service sending the data for the two items into the Oracle system. For both items, the following data is provided:

- Supplier contract number
- Supplier part number
- Supplier unit of measure

UN / SPSC is the category code used for both items.

```
<?xml version='1.0'?>
<OrderLinesDataElements>
<catalogTradingPartner><![CDATA[ABCCatalogServices]]></catalogTradingPartner>

<orderLine>
<contract contractNumberIdentifier='KNOWN'>
    <supplierContract>
        <contractNumber><![CDATA[111111112767-1]]></contractNumber>
        <contractLineNumber><![CDATA[12]]></contractLineNumber>
    </supplierContract>
</contract>

<item lineType='GOODS'>
    <itemNumber>
       <supplierItemNumber>
            <itemID><![CDATA[C13139]]></itemID>
       </supplierItemNumber>
</itemNumber>
```

```
    <itemDescription><![CDATA[Hard drive-540MB IDE]]></itemDescription>
    <quantity><![CDATA[34]]></quantity>
    <unitOfMeasure>
         <supplierUnitOfMeasure>
            <supplierUOMType ><![CDATA[Each]]></supplierUOMType>
            <supplierUOMQuantity><![CDATA[1]]></supplierUOMQuantity>
         </supplierUnitOfMeasure>
    </unitOfMeasure>
</item>

<category categoryCodeIdentifier='SPSC'>
    <categoryCode><![CDATA[5149-9908-00]]></categoryCode>
</category>

<price>
    <currency><![CDATA[USD]]></currency>
    <unitPrice><![CDATA[485.99]]></unitPrice>
</price>

<supplier>
    <supplierName><![CDATA[A-1 Lighting]]></supplierName>
    <supplierSite><![CDATA[Washington]]></supplierSite>
</supplier>
</orderLine>

<orderLine>
<contract contractNumberIdentifier='KNOWN'>
    <supplierContract>
        <contractNumber><![CDATA[222222225678-2]]></contractNumber>
        <contractLineNumber><![CDATA[15]]></contractLineNumber>
    </supplierContract>
</contract>

<item lineType='GOODS'>
    <itemNumber>
       <supplierItemNumber>
            <itemID><![CDATA[P22378]]></itemID>
       </supplierItemNumber>
</itemNumber>
    <itemDescription><![CDATA[High speed assembly machine]]></itemDescription>
    <quantity><![CDATA[9]]></quantity>
    <unitOfMeasure>
         <supplierUnitOfMeasure>
             <supplierUOMType ><![CDATA[Each]]></supplierUOMType>
             <supplierUOMQuantity><![CDATA[1]]></supplierUOMQuantity>
```

```
        </supplierUnitOfMeasure>
    </unitOfMeasure>
</item>

<category categoryCodeIdentifier='SPSC'>
    <categoryCode><![CDATA[5149-9908-00]]></categoryCode>
</category>

<price>
    <currency><![CDATA[USD]]></currency>
    <unitPrice><![CDATA[12575.99]]></unitPrice>
</price>

<supplier>
    <supplierName><![CDATA[JCN Technologies]]></supplierName>
    <supplierSite><![CDATA[New York]]></supplierSite>
</supplier>

</orderLine>
</OrderLinesDataElements>
```

HTML Specification

The HTML format for an external catalog source to transmit the user's selected item data to *i*Procurement in XML is presented below. The format requires fragmenting the XML file into segments.

Sending Selected Item to *i*Procurement: External Catalog's HTML File Format

This is the HTML format used to send selected items from the external catalog to *i*procurement:

```
<HTML>
  <BODY onLoad="document.orderForm.submit()">
  <FORM ACTION="URL of the Web Requisitions" METHOD="POST"
    NAME="orderForm">
    <INPUT type="hidden" name="REQ_TOKEN" value="Requisition token">
    <INPUT type="hidden" name="NO_OF_DATA_SEGMENTS" value="N">
    <INPUT type="hidden" name="ITEM_XML_DATA1" value="First data segment">
    <INPUT type="hidden" name="ITEM_XML_DATA2" value="Second data segment">
........
    <INPUT type="hidden" name="ITEM_XML_DATAN" value="Nth data segment">
  </FORM>
  </BODY>
</HTML>
```

HTML Elements Explained

Table 10–12 lists HTML elements grouped under their corresponding HTML tag names.

Table 10–12 HTML Elements

HTML Element	Format	Description
<BODY>	onLoad="document.orderForm.submit()"	Provides HTTP redirect from client browser to *i*Procurement.
<FORM>	ACTION="*URL of the Web Requisitions*"	Action attribute of the orderForm should be set to the URL that is provided by Self-Service Purchasiing.
	METHOD="POST"	Form submit type.
	NAME="orderForm"	Name of the form that is submitted.
<INPUT>	<INPUT type="hidden" name="REQ_TOKEN" value="*Requisition token*">	Creates a hidden form element for Requisition Token. This token is sent to the 3rd party catalog provider after a successful authentication process and, without change, is sent back to *i*Procurement. It contains internal system-related data.
	<INPUT type="hidden" name="NO_OF_DATA_SEGMENTS" value="*N*">	Creates a hidden form element to send *number of data segments* that is transferred between two systems. The *value N* should contain the number of ITEM_XML_DATA form elements.
	<INPUT type="hidden" name="ITEM_XML_DATA*N*" value="*Nth data segment*">	Creates a hidden form element to transfer a data segment. The data segments are formed by dividing the XML file into smaller portions. The setting for the size of each segment should be a variable. Current recommended value for this variable is 2000 characters. The last character of the name determines the index of the corresponding data segment. The index starts from 1 and is increased by one up to NO_ OF_DATA_SEGMENTS value *N*.

*i*Procurement Example 11: HTML/XML File

This example uses the HTML elements defined in Table 10–12.

```
<HTML>
  <BODY onLoad="document.orderForm.submit()">
  <FORM ACTION="http://ap411sun.us.oracle.com:9999/OA_JAVA_
SERV/wp4/integrate/apps.Order" METHOD="POST" NAME="orderForm">
  <INPUT type="hidden" name="REQ_TOKEN"
value="template=tpn,action=addLines,function=addToOrder,por_req_session_
id=1,.....">
  <INPUT type="hidden" name="NO_OF_DATA_SEGMENTS" value="2">
  <INPUT type="hidden" name="ITEM_XML_DATA1" value="<?xml
version='1.0'?><OrderLinesDataElements><catalogTradingPartner><![CDATA[ABCCatalo
```

```
gServices]]></catalogTradingPartner><orderLine>....."> 
   <INPUT type="hidden" name="ITEM_XML_DATA2"  value="
.....
</orderLine>
  <orderLine>
    <contract
.....
</OrderLinesDataElements>">
</FORM>
</BODY>
</HTML>
```

User Authentication

When you first log into *i*Procurement, a random user identification number for the session is generated (a session "ticket"), encrypted with a one-way encryption (has) and stored in Oracle Procurement Server.

When you select a link for an externally hosted catalog, the encrypted session ticket, and URL for the user's authentication is sent. The following example shows the call that the catalog provider may expect:

```
https://www.extsupplier.com?url=oas.us.oracle.com/wr41102/plsql/icx_ext_
supplier.authenticate_user&ticket=128019274
```

where:

- **www.extsupplier.com** is the URL provided by the external catalog
- **ias.us.oracle.com/wr41102/plsql/icx_ext_supplier.authenticate_user** is the return customer URL (**ias.us.oracle.com** is the URL for the application server located outside the firewall)

The catalog provider then makes an HTTP call to the Procurement Server, using SSL, requesting verification of this encrypted ticket at the URL address sent with the user:

```
https://ias.us.oracle.com/wr41102/plsql/icx_ext_supplier.authenticate_
user?ticket=128019274
```

This is actually a call to a PL/SQL package stored in the database behind the client site firewall.

The external catalog provider makes a connection to an application server outside the firewall, which authenticates the catalog provider's digital certificate. It then allows the call to be made to the internal application server. Here the encrypted session ticket is verified against a stored version in the table, and the rest of the user information is returned to the catalog provider.

*i*Procurement XML Example 12: Valid Session XML Document

If the session ticket is valid, *i*Procurement pass your login back. This includes your name, delivery information, company, operating unit, requisition number, and a return URL to the Requisition Server as illustrated below:

```
<?xml version='1.0'?>
  <RequisitionUser>
   <userName>CBLACK</userName>
```

```
    <company>VIOP</company>
    <operatingUnit>Organization</operatingUnit>
    <shipTo>Philadelphia</shipTo>
    <deliverTo>Philadelphia</deliverTo>
    <reqToken>Req_Token</reqToken>
    <returnURL>ap411sun.us.oracle.com:5555/FJ_JAVA_SERV/faboujaw/PS/tpn_
redirect</returnURL>
  </RequisitionUser>
```

Authenticated XML Schema

When the external catalog provider requests user authentication, *i*Procurement calls a PL/SQL procedure, with a session ticket as the encrypted parameter, to authenticate the user.

*i*Procurement Example 13: Authenticated XML Schema — Returned Requisition User XML Document

The following XML schema shows the returned Requisition User XML document.

```
<?xml version='1.0'?>
<s:schema id='RequisitionUser'>
<elementType id="userName">
     <string/>
     <description>Unique user name of person in the requisition
system</description>
</elementType>
<elementType id="company">
     <string/>
     <description>Unique company name in the requisition system</description>
</elementType>
<elementType id="operatingUnit">
     <string/>
     <description>Unique operating unit name in the requisition
system</description>
</elementType>
<elementType id="shipTo">
     <string/>
     <description>shipTo account for the requisition</description>
</elementType>
<elementType id="deliverTo">
     <string/>
     <description> deliverTo account for the requisition </description>
</elementType>
<elementType id="reqToken">
     <string/>
     <description>Unique requisition ID used in the requisition
system</description>
</elementType>
<elementType id="returnURL">
     <string/>
     <description>URL that ReqLines should be redirected to</description>
</elementType>
```

```
<elementType id="RequisitionUser">
     <elementType ="#userName"/>
     <elementType ="#company"/>
     <elementType ="#organisation" occurs "OPTIONAL"/>
     <elementType ="#reqToken"/>
     <elementType ="#shipTo" occurs "OPTIONAL"/>
     <elementType ="#deliverTo" occurs "OPTIONAL"/>
     <elementType ="#returnURL"/>
     <description>object sent to external supplier for user
 identification</description>
</elementType>
</s:schema>
```

*i*Procurement Example 14: Authenticated User: Sample Returned XML Document

This example has data. It illustrates a typical returned XML document generated for an authenticated user.

```
<?xml version='1.0'?>
<RequisitionUser>
  <userName>Fred Bloggs</userName>
  <company> Oracle Corporation</company>
  <operatingUnit>Manufacturing</operatingUnit>
  <shipTo>Oracle HQ</shipTo>
  <deliverTo>Kevin Miller</deliverTo>
  <reqToken> 1245</reqToken>
  <returnURL>http://reqs.us.oracle.com/order/75</returnURL>
</RequisitionUser>
```

Unauthenticated XML Schema

For the case when the userID sent from the external supplier is not valid, the following schema represents the generated XML:

*i*Procurement Example 15: Unauthenticated User — XML Schema

```
<?xml version='1.0'?>
<s:schema id='InvalidUser>
  <elementType id="message">
  <string/>
  <description>Message generated when an invalid sessionID is sent</description>
  </elementType>
```

*i*Procurement Example 16: Unauthenticated User — Sample XML document

This example illustrates a typical returned XML document generated for an unauthenticated user.

```
<InvalidUser>
 <message>Invalid user</message>
</InvalidUser>
```

11

Customizing Discoverer 3i Viewer with XSL

This chapter contains the following sections:

Discoverer3i Viewer: Overview

XML Components Used: Oracle XML Parser for Java, Version 2

What is Discoverer?

Discoverer Business Intelligence solutions transform an organization's *data* into *information*. Oracle Discoverer for the Web allows you to access this information using a Web browser interface.

What is Discoverer 3i Viewer?

Oracle Discoverer 3i Viewer makes the information available anywhere on the Internet or Intranet, and allows the information to be transparently embedded in Web pages or accessed from corporate Portals. Oracle Discoverer 3i Viewer can be customized, to fit any Web site using standard Web Technologies such as XML and XSL.

Discoverer allows you to make queries, while Reports lets you publish reports in different formats, including HTML, Adobe's Portable Document Format (PDF), and XML.

Customizing Oracle Discoverer™ 3i Viewer

This chapter provides customization examples and describes strategies for using Discoverer 3i Viewer.

- XML and XSL: Discoverer3i Viewer uses industry standard XML to represent data and application states, and the XSL style sheet language to format the User Interface. Standard XSL tools can be used to customize the User Interface or to produce a complete embedded Business Intelligence application.
- HTML: You can specify HTML formatting attributes in a single customization file. Fonts, colors, and graphics are easily changed especially if you are familiar with HTML formatting.

Discoverer3i Viewer can be driven and accessed by middle-tier B2B applications.

More Information on Discoverer 3i Viewer

For more information on Discoverer3i Viewer see:

http://technet.oracle.com/docs/products/discoverer/doc_index.htm

Discoverer 3i Viewer: Features

Discoverer 3i Viewer allows you to:

- *Run Reports*
  - Dynamically run reports saved in the database.
  - Enter parameters at runtime.
  - Run scheduled reports.
  - Cancel a query.
  - Page between dimension values on the page axis.
  - Change workbook and database options.
  - Print reports.
  - Export reports to various file formats such as HTML, Excel, and other PC file formats.
- *Analyse Data*
  - Perform drill down analysis.
  - Drill through different levels of summarized data.
  - Drill out to data held in other applications, such as web pages, MS Word documents etc.
- *Control Queries*
  - Control query execution time. If a query is still running when the time threshold is reached, the query is automatically terminated.
  - Display a query estimate. If a query is predicted to take longer than a predefined time threshold, Discoverer Viewer warns you and allows you to determine if the query should be run.
  - Automatically redirect queries to summary tables. Requests for summarized data are automatically redirected to a summary table containing pre-summarized data.
- *Secure Data Access*
  - Leverage the security features of the web server and database.
  - Go through multiple firewalls
  - Support SSL, x.509 and other standard web security protocols

    - Support disconnected access to the data
- *Use Browser Options to:*
    - Bookmark favorite reports
    - Embed reports in other web pages
    - Change font sizes and link styles by changing browser options
    - Export output to other formats, such as Excel for further spreadsheet analysis.
    - View reports offline (Internet Explorer 5)

  Discoverer 3i Viewer uses no Java, no Javascript, and no frames, enabling even low specification browsers to be used.
- Discoverer reports can be embedded:
    - In existing Web Pages by specifying a URL that defines the workbook and worksheet to be included. When the link is clicked the database is queried and the latest data is displayed in HTML.
    - In portals such as Oracle Portal (also known as *i*Portal and previously known as WebDB) and can take on the look and feel of the hosting portal.
- Used to build complete custom Web applications or deliver data to other middle tier web systems.

Discoverer 3*i* Viewer can be used in the following ways:

- As a standalone Business Intelligence tool
- To integrate database output into your Web site and portal
- Customized to fit in with your Web site look and feel, to incorporate your companies logo or other artwork, or to build custom Discoverer applications for the Web.

Discoverer 3*i* Viewer: Architecture

The Discoverer 3*i* Viewer architecture is shown in Figure 11–1.

Discoverer 3*i* Viewer components are listed below:

- Oracle Discoverer Application Server: The engine for Discoverer Web solutions
- Web Server and servlet container, such as, Apache and Apache JServ (JVM)
- Oracle XML Parser for Java v2. This includes the XSL-T Processor

- Discoverer 3*i* Viewer Servlet
- Discoverer Server interface, a Java module
- Oracle8*i* database

How Discoverer 3*i* Viewer Works

See Figure 11–1 to understand how Discoverer 3i Viewer works:

1. Discoverer 3i Viewer is invoked via a URL from a standard Web Browser, just like any other Web Site. The URL is processed by the 3i Viewer Servlet running on the Web Server.

   The servlet uses Discoverer Server Interface (Model) to communicate with the Discoverer Application Server. Discoverer Server Interface and Discoverer Application Server are both also used by Discoverer User Edition:

   Discoverer Server Interface. This is an applet but here it is running on the Web Server, rather than in the client's JVM as in Discoverer User Edition. The 3i Viewer Servlet communicates with Discoverer Application Server using Corba IIOP protocol.

   Discoverer 3i Viewer andDiscoverer User Edition use the same Discoverer Application Server.

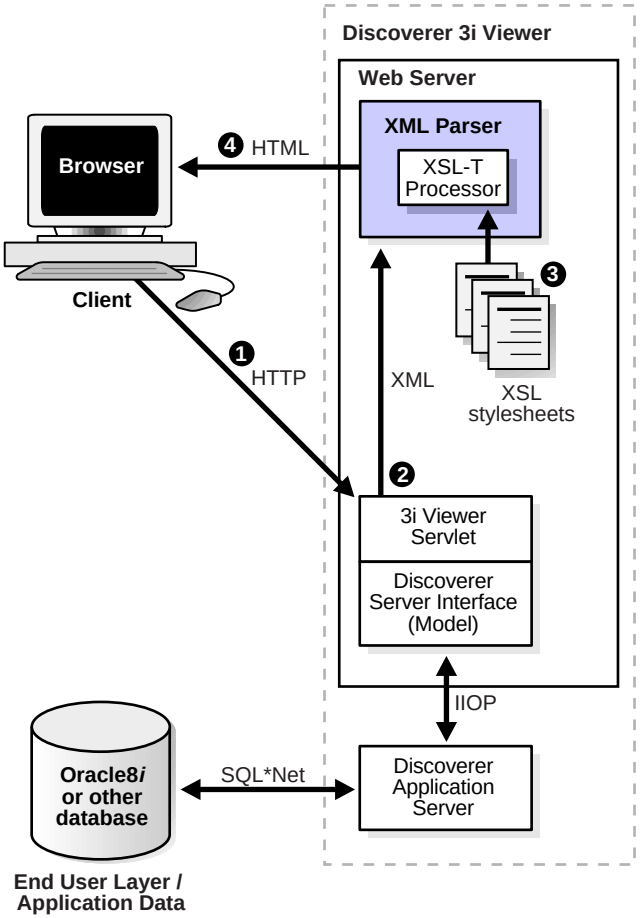
2. Discoverer 3i Viewer Servlet interprets the HTTP request from the client browser, and makes the necessary calls to the Discoverer Application Server. The server response is represented in XML generated by the servlet and is sent to the XML / XSL processor (XSL-T Processor).

3. This combines the XML with an XSL configuration file that defines the representation of the User Interface and,

4. XSL-T Processor generates HTML to send back to the browser.

   It is the XSL file that allows the User Interface of Discoverer 3*i* Viewer to be customized for individual sites.

Figure 11–1 Discoverer 3i Viewer Architecture



Replicating Discoverer Application Server

The Web Server and the Discoverer 3*i* Viewer Servlet container can be replicated using standard web farming and virtual hosting techniques.

In a real system there would be many users using each web and application server. Discoverer allows you to determine exactly how you want the load spread across available machines.

Using Discoverer 3*i* Viewer for Customized Web Applications

Discoverer 3*i* Viewer generates HTML by using the following XML components:

- XML, which describes the information available
- XSLT Processor and XSL stylesheets which define how that information should be represented in HTML

XSL configuration file (stylesheet) defines simple attributes, such as the fonts and colors to use, but it also defines the layout of each page, and the interactions with the user. By customizing the XSL stylesheet, specific Discoverer applications can be built and delivered on the Web.

Note: The application described here was run on Internet Explorer 5.x browser.

Step 1: Browser Sends URL

After login, assume a Discoverer Viewer has asked for a list of workbooks that these workbooks are allowed to be opened in order to analyse their business. The URL issued is http://ukp14910.uk.oracle.com/disco/disco3iv?us=video&db=Disco

The URL specifies the machine the servlets are installed on, the username, and database connection string to use. The password is not normally shown on the URL for security reasons.

Step 2: Servlet Generates XML

Discoverer 3*i* Viewer Servlet processes the URL. It instructs the Discoverer Application Server to check the security setting for this user and return details of the workbooks that this user is allowed to access.

Security settings are held in the End User Layer tables in the database. After this information is returned from the Discoverer Application Server, the servlet generates the following XML in which you can see information about the three workbooks being returned:

- Store and Band Analysis workbook
- Video Sales Analysis workbook
- Annual Sales Report workbook

Discoverer XML Example 1: Three Workbook Report Data

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="example1.xsl"?>
<discoverer version="3.5.8.12" login_method="discoverer">
 <account name="myname@mydatabase" ref="MYNAME%40mydatabase">
  <user>MYNAME</user>
   <database>mydatabase</database>
   <eul default="true" name="myeul">
   <workbook name="Store and Band Analysis" ref="Store~20and~20Band~20Analysis">
      <description>Shows sales by Store, broken into sales bands</description>
   </workbook>
   <workbook name="Video Sales Analysis" ref="Video~20Sales~20Analysis">
      <description>General purpose analysis of the business</description>
   </workbook>
   <workbook name="Annual Sales Report" ref="Annual~20Sales~20Report">
      <description>Shows yearly and quarterly sales of products</description>
   </workbook>
   </eul>
 </account>
</discoverer>
```

Note: Tere is no information in the XML about how these workbooks names and descriptions *should be displayed* to the user. This is the function of the XSL file.

Step 3: XSL-T Processor Applies an XSL Stylesheet

XSL is the industry standard stylesheet language defined by W3C. It allows a selection of elements from an XML file to be combined with an HTML template to generate HTML output for a Web Browser.

Discoverer 3i Viewer User Interface is entirely defined in XSL. This means it can be customized or copied to define alternative User Interface (UI) styles using standard Web development tools, such as HTML editors, XSL editors, or even simple text editors.

Step 4: XSL-T Processor Generates HTML

The XSL and XML .

Using the XML generated in Step 2 and the standard Discoverer 3i Viewer XSL configuration file (stylesheet), these are combined in the XSL-T processor in the

XML Parser for Java,v2. This then generates the HTML version of the XML document.

This HTML is sent back to the browser in response to the initial URL.

In Discoverer 3i Viewer, the generated HTML does not use frames or Javascript, and therefore makes minimal demands on the browser or internet device. Hence it is easy to integrate with other web applications or portals.

Customizing Style by Modifying an XSL Stylesheet File: style.xsl

You need to be able to easily modify fonts and colors to fit in with your corporate standards, or to display the company logo to add branding. These global changes can be made in a single XSL stylesheet file "style.xsl" that defines special 'tags' for each style that can be modified. For example:

- *Inserting Logos*: To insert a logo change the following line :

  ```
  <xsl:variable name="logo_src"> </xsl:variable name>
  ```

  to

  ```
  <xsl:variable name="logo_src"> http:www.mycompany.com/images/mylogo.gif
  </xsl:variable name>
  ```

- *Changing the Text Color*: To change the color of the text, change the following line and add the appropriate color code.

  ```
  <xsl:variable name="text_color">#000000</xsl:variable>
  ```

Many global style changes can be made in this way, but the overall operation of the User Interface remains unchanged. This is only one way of customizing Discoverer 3i Viewer. In fact, using XSL allows a complete customized application to be made, as the next example shows.

Discoverer 3i Viewer: Customization Example Using XML and XSL

You can use the XML and XSL fragments below to experiment with customization in a Web Browser.

Step 1: The XML File

The data is a standard XML file, similar to the previous "Discoverer XML Example 1":

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="example1.xsl"?>
<discoverer version="3.5.8.12" login_method="discoverer">
 <account name="myname@mydatabase" ref="MYNAME%40mydatabase">
  <user>MYNAME</user>
   <database>mydatabase</database>
   <eul default="true" name="myeul">
   <workbook name="Store and Band Analysis" ref="Store~20and~20Band~20Analysis">
      <description>Shows sales by Store, broken into sales bands</description>
```

```
   </workbook>
   <workbook name="Video Sales Analysis" ref="Video~20Sales~20Analysis">
      <description>General purpose analysis of the business</description>
   </workbook>
   <workbook name="Annual Sales Report" ref="Annual~20Sales~20Report">
      <description>Shows yearly and quarterly sales of products</description>
   </workbook>
   </eul>
 </account>
</discoverer>
```

The XML file starts by specifying the XML version. The 2nd line specifies the XSL file to be applied to process the data, "example1.xsl" and the rest of the file is generated from the Discoverer 3i Viewer.

The first two lines have been added here so that you can type the text into a file using a text editor and then open it in a Web Browser to see the results visually as the XSL is changed. Save the file with the extension "xml" if you want to try this.

Step 2: XSL File, example1.xsl

XSL file, "example1.xsl", looks like this :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
  <html>
    <body bgcolor="#ffffff" link="#663300" text="#000000">
       <b><i>Choose a Workbook:</i></b>
       <br/>
       <table border="2">
         <xsl:for-each select="/discoverer/account/eul/workbook">
          <tr>
            <td width="242">
              <font face="sans-serif">
                <xsl:value-of select="@name"/>
              </font>
            </td>
            <td>
               <xsl:value-of select="description"/>
            </td>
          </tr>
        </xsl:for-each>
       </table>
    </body>
```

```
  </html>
</xsl:template>
</xsl:stylesheet>
```

Step 3: XML+XSL = HTML

Figure 11–2 shows what you see on a Browser when the XML file is opened in the Browser, the Browser reads in the XSL stylesheet (example1.xsl), and generates HTML.

Figure 11–2 List of Workbooks Viewed on Browser, XML+ example1.xsl=HTML — Before Modification

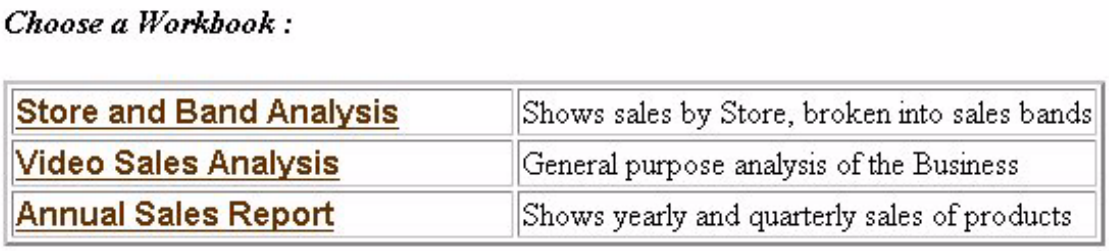


Table 11–1 examines the XSL file, example1.xsl, from line 5. It describes how the HTML is generated. The file starts by specifying the XML version. The 2nd line says that this file is a stylesheet. The HTML template starts with the <HTML> tag on line 4.

Table 11–1 Explaining example1.xsl — Before Modifying the XSL File

example1.xsl code	The code means ...
<body bgcolor="#ffffff" link="#663300" text="#000000">	This line defines the colors to be used
Choose a Workbook :	This is just more HTML. It sets a bold italic font and inserts the text "Choose a workbook :"
<table border="2">	An HTML table is started, with a 2 line border.

Table 11–1 Explaining example1.xsl (Cont.) — Before Modifying the XSL File (Cont.)

example1.xsl code	The code means ...
<xsl:for-each select="discoverer/account/eul/workbook">	This is the first real XSL code. It means : Go through the XML data file and for each workbookinfo tag perform all the following steps until you reach the end tag : </xsl:for-each>. So for every workbook that appears in the XML file the following XSL is processed, and a row is inserted into the HTML table for every workbook found :
<tr> <td width="242"> <xsl:value-of select="@name"/> </td>	<tr> starts a new row in the table <td> defines the table data to be inserted for the first column. The width of the column is set to 242 pixels and the font is set to sans-serif. The XSL line inserts the text from the XML file for the <NAME> tag under each workbookinfo section.
<td> <xsl:value-of select="description"/> </td> </tr>	These lines define the 2nd column in the HTML table and insert the text for the workbook description using the <DESCRIPTION> tab in the XML file. So each row in the HTML table will contain the workbook name, made into a link to click on, and the workbook description as text. Since there are three workbooks in the XML file, there will be three rows in the table.

Note:

- This example is not *exactly* how the Discoverer 3*i* Viewer shows the list of workbooks. It has been simplified here for clarity, but it illustrates how the XSL stylesheet controls the appearance of the output. See Figure 11–4 for a more typical rendition.
- In Discoverer 3*i* Viewer, the XML and XSL are combined in the XSL-T Processor on the middle tier, and not in the Web Browser.

Step 4: Customizing the XSL Stylesheet (example2.xsl)

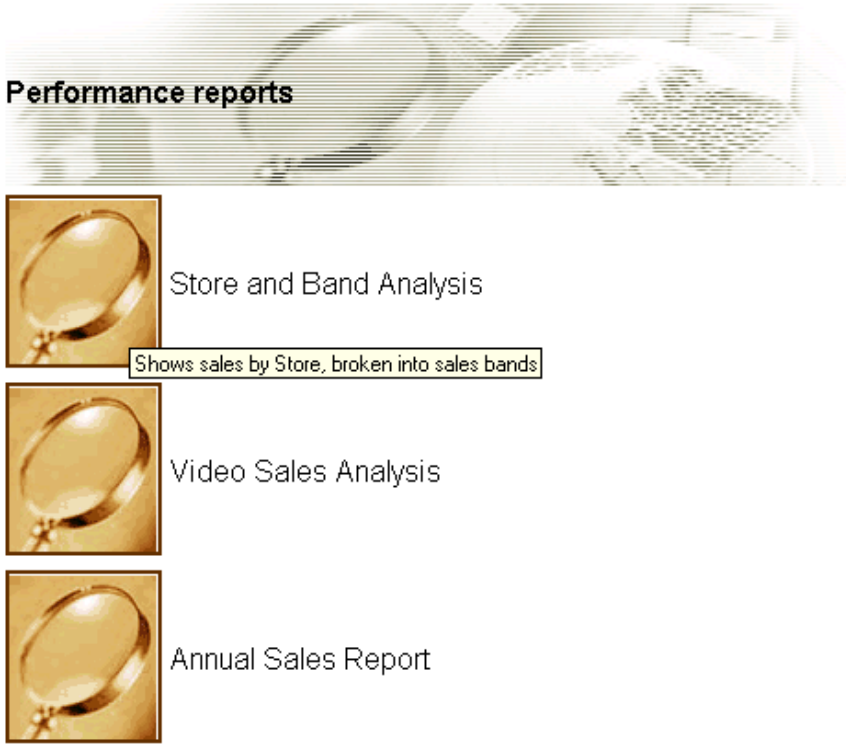
The XSL stylesheet is modified as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/TR/WD-xsl">
```

```
<xsl:template match="/">
 <html>
  <body bgcolor="#ffffff" link="#663300" text="#000000">
   <table border="0">
      <tr>
        <td>width="500" height="100" background="disco_banner.gif">
          <font face="sans-serif">
            <b>Performance Reports</b>
          </font>
        </td>
      </tr>
   </table>
   <table border="0">
      <xsl:for-each select="/discoverer/account/eul/workbook">
         <tr>
           <td>
             <font face="sans-serif">
               <b>
                 <a href="link.htm">
                    <img src="button2.gif">
                      <xsl:attribute name="alt">
                      <xsl:value-of select="description"/>
                      </xsl:attribute>
                    </img>
                 </a>
               </b>
             </font>
           </td>
           <td>
             <font face="sans-serif">
                <xsl:value-of select="@name"/>
             </font>
           </td>
         </tr>
      </xsl:for-each>
   </table>
  </body>
</html>
</xsl:template>
</xsl:stylesheet>
```

When this is combined with the same XML, it appears as shown in Figure 11–3.

Figure 11–3 Displayed Workbook List Using Same XML with a Modified XSL Stylesheet



Now the appearance of the User Interface is completely different, as it takes on a more graphical look and feel. Instead of text links there are graphical buttons for running the reports, each with a dynamic 'tool tip' that pops up when you position the mouse over the button.

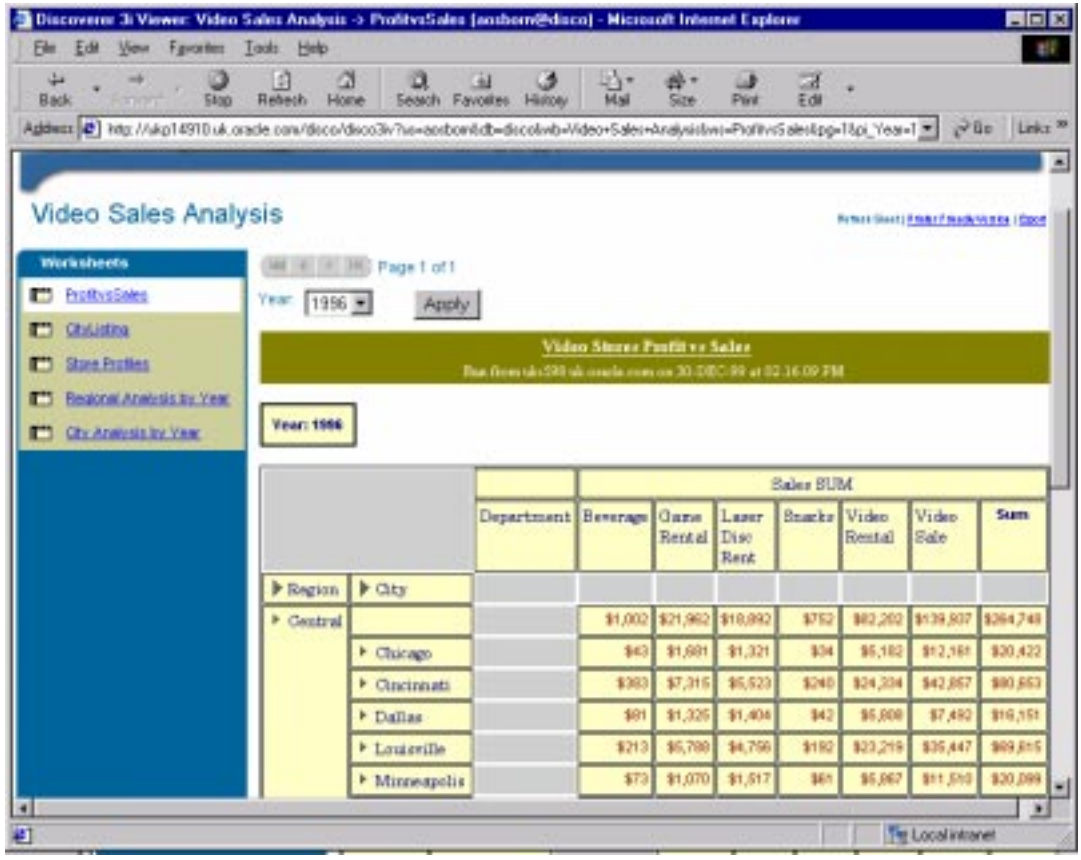
The modified XSL file is described in Table 11–2.

Figure 11–4 shows a typical web-based rendition of this sample application.

Table 11–2 Explaining example2.xsl — After Modifying the XSL File

example2.xsl code	The code means ...
<table border="0"> <tr> <td width="500" height="100" background="disco_banner.gif"> Performance reports </td> </tr> </table>	These lines create a table and insert a graphic and the heading "Performance Reports"
<table border="0"> <xsl:for-each select="discoverer/account/eul/workbook">	This starts the main table that the workbook names will be displayed in, as before, but now there is no border around the table and the rows are defined differently:
<tr> <td> <img src="button2.gif"> <xsl:attribute name="alt"> <xsl:value-of select="description"/> </xsl:attribute> </img> 	The first table data column is defined as a hyperlink again, but this time with the image "button2.gif" as an image, rather than a text link. The font used is "sans-serif". To get a "tooltip" to appear over an image the HTML "alt" attribute is used. Normally the alt attribute is used with a simple text string : <img src="button2.gif" alt="Tooltip text to appear when a mouse is over the image"> but since we want the tool tip to be dynamic we generate the alt tag by getting the text from the <description> tab in the XML file. The <xsl:attribute> tag does this. <xsl:value-of select="description"/> The second data column selects the name of the workbook to display, by using XSL to get it from the XML file as before.

Figure 11–4 Discoverer 3i Viewer. Typical Web-Based Rendition as a Business Solution



Frequently Asked Questions (FAQs): Discoverer 3i Viewer

Explaining Servlets

Question

What is a servlet?

Answer

Servlets are modules of Java code that run in a server application (hence the name "Servlets", similar to "Applets" on the client side) to answer client requests. Servlets are not tied to a specific client-server protocol but they are most commonly used with HTTP and the word "Servlet" is often used in the meaning of "HTTP Servlet".

Servlets make use of the Java standard extension classes in the packages javax.servlet (the basic Servlet framework) and javax.servlet.http (extensions of the Servlet framework for Servlets that answer HTTP requests). Since Servlets are written in the highly portable Java language and follow a standard framework, they provide a means to create sophisticated server extensions in a server and operating system independent way.

Typical uses for HTTP Servlets include:

- Processing and storing data submitted by an HTML form.
- Providing dynamic content, such as returning the results of a database query to the client.
- Managing state information on top of the stateless HTTP, such as for an online shopping cart system which manages shopping carts for many concurrent customers and maps every request to the right customer.

How Discoverer 3*i* Viewer Communicates with Browsers

Question

What does Discoverer 3I Viewer use to communicate with the user's browser?

Answer

HTTP and HTML.

Why HTML is Output to the Browser

Question

Why does Discoverer 3i Viewer only output HTML to the user's browser?

Answer

Discoverer 3i Viewer outputs 100% HTML so that it can support the widest possible range of browsers. Using this architecture also avoids the need for an end user to perform an install or download.

Discoverer 3i Viewer and XML

Question

How is XML used by Discoverer 3I Viewer?

Answer

XML is generated by the middle-tier and represents the application state. Discoverer 3iViewer Servlet interprets an HTTP request from the user's browser, and makes the necessary calls to the Discoverer Server.

The server response is represented in XML generated by the Servlet. XSL is applied to this XML, producing the HTML that is displayed by the users browser.

By using XML and XSL together, the underlying data and the look and feel are separated allowing easy customization.

disco3iv.xml

Question

What does the disco3iv.xml file do?

Answer

You can use disco3iv.xml file to configure various options to make Discoverer 3i Viewer behave the way you want to. For example, you can specify the Discoverer Session that it should connect to.

XSL

Discoverer 3i and XSL

Question

How is XSL used by Discoverer 3i Viewer?

Answer

Discoverer 3I Viewer uses XSL (or more specifically XSL-T) to transform the XML generated by the middle-tier into the HTML that is sent to the user's browser. By editing the XSL files, you gain complete control over the style and presentation of the UI.

Supported XSL-T Processors

Question

What XSL processors can be used by Discoverer 3i Viewer?

Answer

Discoverer 3i Viewer can be configured to use:

- Oracle XSL -TProcessor (default, part of the XML Processor for Java)
- James Clark's XT (not supplied)

Specifying the XSL-T Processor in the Servlet's Classpath

Question

How can you specify which XSL processor Discoverer 3i Viewer uses?

Answer

By default Discoverer 3i Viewer uses Oracle XSL-T Processor. If you prefer you can use James Clark's XT. The Servlet can be reconfigured to use the James Clark XSL processor as follows:

1. Include the XT processor in the servlet's classpath. The XT processor is available as a Zip file, xt.zip, from http://www.jclark.com/xml/xt.html
2. Unpack the Zip file to access the xt.jar file

The JAR should be included in either the class path of the Servlet engine or the classpath for the Servlet engine zone. Different servlet engines have different ways of doing this.

For example, for Apache JServ the servlet engine's class path is setup in the wrapper.classpath entry of the JServ.properties file. The classpath for the servlet zone is specified as a repository in the servlet zone properties.

For details on how to set up each servlet engine read the section in the *Discoverer Installation and Administration Guide* on how Visibroker jar files are installed for that servlet engine. The process for installing the XT processor Jar file is identical.

3. Modify the Discoverer Viewer configuration file. Include the following line should in the <document> element of the disco3iv.xml configuration file:

```
<argument name="xsl_processor">com.jclark.xsl.sax.XSLProcessor</argument>
```

4. Restart the web server.

XSL Editors

Question

What tools are available to edit XSL Stylesheets?

Answer

You can use any text editor to edit XSL files however the following applications are designed especially for editing XSL:

- eXcelon Stylus
- IBM XSL Editor

Customizing Stylesheets

Question

What is commonly changed in order to customize a stylesheet?

Answer

To customize a styelsheet, edit the following items:

- **disco3iv.xsl** to define the types of HTML pages and the rules for when they are displayed
- **page_layouts.xsl** to define the overall layout of each type of HTML page
- **gui_components.xsl** to define the look of each GUI component used in the page layouts
- **style.xsl** to define the style of various fonts used in Discoverer 3i Viewer
- **errors.xsl** to create your own custom error messages
- **functions.xsl** (not recommended) to change the behavior of the core functions used by the other XSL files

Viewing Changes to a Modified Stylesheet

Question

When I customize my own XSL Stylesheet, why can't I see my changes?

Answer

By default, the XSL-T Processor caches the XSL files in its memory for better performance. You have two options for viewing the changes:

- Restart the servlet (by restarting the web server) every time you want to see your new changes. This makes the servlet re-read the XSL files from disk.
- Disable XSL caching. To do this, add the following line to the <document> section of the disco3iv.xml file and restart the web server.

```
<argument name="xsl_cache">false</argument>
```

Browser Displays Blank Screen

Question

Why does my browser display a blank screen?

Answer

This is usually because you have done either of the following:

- Called an XSL template that does not exist

- Tried to use a variable that does not exist

More information on XML and XSL

Question

Where can I find more information on XML and XSL?

Answer

- http://java.sun.com/docs/books/tutorial/servlets/overview/index.html
- http://www.w3.org/Style/XSL/
- http://www.w3.org/XML/
- http://www.builder.com/Authoring/XmlSpot/?tag=st.cn.sr1.ssr.bl_xml
- http://zvon.vscht.cz/HTMLonly/XSLTutorial/Books/Book1/bookInOne.html
- http://www.arbortext.com/Think_Tank/Norm_s_Column/Issue_One/Issue_One.html

12

Phone Number Portability Using XML Messaging

This chapter introduces you to the Phone Number Portability application that uses XML as the message payload.

This chapter contains the following sections:

Introduction to Phone Number Portability Messaging

This chapter provides an overview of the Phone Number Portability message based product, referred to here as Number Portability.

Number Portability is a mechanism by which consumers can keep their telephone numbers when they switch between telecommunication service providers, move from one physical location to another or change their services. The concept is driven by regulatory authorities working to jump start competition, citing that consumers are more interested in moving between service providers when they can keep their telephone numbers. Number Portability is widely cited as a key driver for the explosive growth in the US competitive long distance market.

The Number Portability message-based application uses *i*Message Studio, Event Manager, and Adapters. The application uses XML as the message payload to communicate between two service providers using a Business-to-Business protocol that is common in the telecommunications industry.

It illustrates the messaging and event management features of the Oracle Service Delivery Platform (OSDP or SDP) in Oracle CRM Applications 11i. This is a CRM feature.

Number Portability Allows Fast Configuring

The Number Portability product allows consultants to:

- Implement the product quickly by configuring an XML message DTD in the application
- Be able to assign the different nodes of the XML message to an Oracle data source, an SQL query or a stored procedure.
- Nest SQL queries

For example, to get list of depts and all emps in each dept in an XML message can be performed by doing the following:

1. Writing two queries in the Number Portability application
2. Configuring the message in the supplied GUI with no coding at all

Advanced Queueing in Number Portability will use XML messages as a standard format for communication between the database and external system adapters.

What are External Adapters?

External adapters are Java programs running "listening" to the following:

- Database pipe for commands to perform
- Advanced Queuing (AQ) for messages to process (in multiple threads)

The commands are sent in XML format to the method, `performControlMessageProcessing`. This allows for a dynamic number of parameters to be passed to the adapter.

For example, to start an adapter up with a default of three threads for performance, the STARTUP command could be as follows:

```
<COMMAND>
   <MESSAGE_CODE>STARTUP</MESSAGE_CODE>
   <INITIAL_THREADS>5</INITIAL_THREADS>
</COMMAND>
```

This gives more control and flexibility to you if you need to customize adapters. You can also define your own commands. You are not restricted in any way when parsing XML messages.

See Also:

- *PL/SQL User's Guide and Reference*
- *"Oracle Number Portability 11i User's Guide"* for information on the user interface and iMessage Studio.
- *Implementing Oracle SDP Number Portability*

Terms Used in This Chapter

The following defines terms used in this chapter:

- NPAC.Number Portability Administration Center
- NRC. Number Registration Center. Another name for NPAC.
- SDP. Oracle Service Delivery Platform (SDP)

Requirements for Building a Phone Number Portability Application

To build a Number Portability application, you need the following:

- Oracle Applications 11*i*
- Oracle SDP Number Portability 11i Release 2
- Oracle8*i*

Number Portability and Messaging Architecture within SDP

The Number Portability and messaging architecture in the Oracle Service Delivery Platform (SDP) framework comprises the following components. Event Manager is the core component.

- Communication Protocol Adapter
- Order Processing Engine
- Workflow Engine
- Fulfillment Engine
- Event Manager
- SDP Repository

Communication Protocol Adapter

Communication Protocol Adapter interfaces between SDP and external systems. It handles the following message flow:

- Incoming orders are taken by the appropriate Communication Protocol Adapter and passed to SDP.
- Out going messages are passed from SDP to an external system.

It supports the following adapters:

- File / FTP. This supports a batch mode processing.
- HTTP
- Script
- Interactive Adapter, such as, Telnet sessions

Order Processing Enginer

Orders from an order management system are converted to SDP Work Items or logical line items. Orders are also created internally from processing the messages.

These line items created are analyzed by Dependency Manager or Order Analyzer. It creates a final set of Normalized Work Items for execution.

Workflow Engine

This module specifies the actual flow of actions (known as Fulfillment Actions) to be executed to satisfy an application functionality. This module can re-use the Fulfillment Actions to customize any new functionality such as NP Service Provider Mediation or the NRC itself.

The Workflow Engine would determine the Fulfillment Actions to execute for each Work Item and determine the Network Elements that it would need to talk to. The Workflow engine also picks and executes an appropriate fulfillment procedure based on the fulfillment element type, software version of the fulfillment element type and the adapter type.

The fulfillment procedures then use the Internet Message Studio generated code to send and process messages. Once it gets an event notification of the outcome of the execution of the Fulfillment Action (by the Event Manager), the engine would proceed to complete the Work Item and pick the next Work Item in the queue for the given order. This component uses the Oracle Workflow engine.

Fulfillment Engine

The Fulfillment Actions and the Network Elements on which they need to be applied are used by SDP's provisioning engine to determine which Fulfillment Program to execute.This essentially uses the PL/SQL engine in the database currently to execute user defined procedures.

Event Manager

The Event Manager is a generic Publish-Subscribe module which registers interest of various subscribers to different event types. The subscriber could be the SDP Translator (in which case the event gets propagated as a new order), Workflow Engine (in which case the event restarts a Workflow which is waiting on an external event) or an API. Event Manager builds asynchronous application messaging. It has a versatile set of API's which can be used by a developer to build asynchronous message based application.

SDP Repository

The core SDP repository allows the user to create orders and configure network elements. An example would be Work Item, Fulfillment Action, Fulfillment Program and Network Element definitions. The NP database contains entities for storing NP specific data such as, Subscription Version, Service Providers, Routing Numbers,...

The Number Portability Process

Number Portability performs the following tasks:

1. Links the XML or DTD elements to either a SQL table or a PL/SQL stored function.
2. Dynamically creates and builds the stored procedure in the background. It also enqueues the XML message for further processing. It builds the XML by extracting/querying values from the table or by dynamically executing the stored function associated with the element.
3. At run time the user or program executes this dynamically executed procedure which then has the intelligence to create the XML message and enqueue it for further processing.

Number Portability product serves as a work flow manager. It is used for provisioning services requested by customers.

What Happens Behind the Scenes When You Order a New Telephone Service

For example, when you order a new telephone service, the telephone company takes the order and captures the order information using the Oracle Order Management application.

Here is the flow of events that transpire. The Number Portability application is used in all of the following steps and serves as an instance in the process:

1. A customer places an order for a service such as a new telephone installation
2. The Provisioning application captures the sales order and starts the specified validating and authorization process
3. The Provisioning application then communicates with external systems. For example, it checks the customer credit rating or interacts with a third party for other actions.

   This communication could use a protocol defined between the two systems in XML format. Oracle Work Flow is used so that consultants can configure and view business process in a graphical format even at runtime.

What Happens Behind the Scenes When You Change Local Service Providers

Number Portability is also in action when you switch local telephone service providers. Here is the process:

1. A customer contacts the local service provider.

2. The local service provider validates your request with your old service provider. This is done through an independent third mediating party. In the United States, this third party mediator is the NPAC (Number Portability Administration Center).
3. When switching long distance carriers, the mediating party comes online via voice. When switching local provider service, this is done through electronic messaging.
4. The new service provider sends a message to NPAC so that it NPAC can validate the request and then an approval or authorization can be granted to the new service provider so that they can gain this new customer.
5. NPAC sends a message to the old provider ("donor"). The donor reviews and approves the order and sends a message back to NPAC again using XML.
6. NPAC sends the authorization to the new service provider ("recipient").
7. The order is now approved on both sides.

> **Note:** All the messaging taking place here uses XML as the main format within the SDP Number Portability product. If another protocol is required, then a custom Adapter could be plugged in to perform the transformation using XSL or custom code.

8. On the actual day that the customer wants to switch, NPAC sends a broadcast message to all the telephone service providers throughout the country. At this time, all telephone carriers and companies must update their network elements (network databases) in the process within the system.

XML is the Data Format. Advanced Queuing is Used at Each Point

XML is the data format used for all the messaging. Advanced Queueing (AQ) is used at each point in the process (and system).

The "Message Builder" module creates and enqueues the XML messages. The Communication Protocol Adapter ("Adapter") starts dequeueing the messages and sends them to the external systems.

AQ is essentially used simply to store the messages in queues. It serves as a First In First Out (FIFO) queueing system. The protocol used to send the messages differs with every system and is end-user specified, such as Flat File / CORBA, ...

To summarize then:

- When an order is received to switch local or long distance carriers, the order request is sent as XML or transformed messages to NPAC
- On approval and authorization to make the change, NPAC sends an XML message to all telephone carriers throughout the country. These carriers then provision their own network elements.

Why XML is Used for this Messaging

XML is used because it is a flexible format that can be modified or transformed into any other format required.

For example, one country may need a flat file message format to distribute the messages and provision (update) their network elements (databases). It is a simple matter to use XSL or custom code to transform the generated XML into the required flat file format.

This Number Portability application has been successfully deployed in Belgium where it is used in this manner. Belgium requires a flat file message format.

Provisioning a Network Element

There are other network elements in the telephone system that can be provisioned (updated) besides individual end-user phone numbers. Examples of other network elements, include switches, Service Control Points (SCP), routers, LDAP servers,....

Here is another example of how the SDP Provisioning application is used:

1. A local telephone service provider requests to have a switch provisioned
2. A Mediation Layer talks to the switches
3. Service Delivery Platform (SDP) which may be an XML-enabled legacy system, sends a message to the Mediation Layer.
4. SDP receives a response back from the Mediation Layer once the provisioning (updating) has completed.

This messaging also uses XML as the message payload and Advanced Queueing (AQ). Here AQ is used mainly as storage medium for the XML queues. Future releases may use the JMS interface over AQ to provide a standard interface.

Using Event Manager to Send and Receive Messages Asynchronously

SDP can send and receive messages asynchronously using Event Manager. Table 12–1 lists the queues and services in SDP that implement Event Management.

Table 12–1 SDP Queues and Services that Implement Event Management

Queue Name	Service Name	Remarks
Inbound Message Queue	Message Server	Processes all incoming messages.
Internal Events Queue	Event Server	Events generated for internal consumption are enqueued on the internal events queue for speedy processing.
Outbound Message Queue	Communication Adapters	Dequeues messages from the Outbound Message queue and passes it to the peer system.

The Event Manager handles all messages entering the application system. Messages coming into the system can be responses to request messages or event notifications from remote systems.

Example Code to Send Messages

Here is an example code fragment that constructs a PORT_IN message and sends it to the NRC consumer. The NRC consumer is a SDP adapter that delivers the message to the NRC system.

```
DECLARE
l_error_codeNUMBER ;
l_error_messageVARCHAR2(2000) ;
l_msg_headerXNP_MESSAGE.MSG_HEADER_REC_TYPE ;
l_msg_textVARCHAR2(4000) ;
l_fnd_messageVARCHAR2(4000) ;
BEGIN
/*
Create a PORT_IN request message
*/
PORT_IN.CREATE_MSG(XNP$TN=>'3037505639'
  XNP$PORTING_ID=>1001,
  x_msg_header=>l_msg_header,
    x_msg_text=>l_msg_text,
       x_error_code=>l_error_code,
      x_error_message=>l_error_message,
        p_sender_name=>'TELIA') ;
     /*
Notify the customer care system and get concurrence
before sending the message to NRC
*/
IF (l_error_code = 0) THEN

/*
Custom procedure to notify the customer care system
*/
NOTIFY_CUSTOMER_CARE(l_msg_header,
l_msg_text,
l_error_code,
l_error_message) ;
IF (l_error_code = 0) THEN
 XNP_MESSAGE.PUSH(P_MSG_HEADER=>l_msg_header,
  P_BODY_TEXT=>l_msg_text,
  P_QUEUE_NAME=>XNP_EVENT.C_OUTBOUND_MSG_Q,
  P_RECIPIENT_LIST=>'NRC_ADAPTER' );
...
```

Messages can also be enqueued on the inbound messages queue or the internal events queue, however these queues are single consumer queues. Each queue can be shared by applications using correlation identifiers.

Using Internet Message Studio (*i*Message) to Create an Application Message Set

Internet Message Studio (iMessage) utility is used to define the message set of the Number Portability application or enterprise. It provides for an easy way to develop a message based application and generates all the necessary code to construct, publish, validate and process application messages.

It also enables sharing of messages between applications and prevents redefining the same message in various applications across the enterprise. The application can execute the generated procedures at run time for all its messaging needs. It also provides the necessary hooks or customization points for including business specific logic. Messages are generated using standard XML.

Code Generation

For every message defined, the *i*Message creates a package with the name of the message and the following procedures as part of the package.

- CREATE_MSG()
- SEND()
- PUBLISH()
- VALIDATE()
- PROCESS()
- DEFAULT_PROCESS()

Defining Message Sets

Figure 12–1, "Using iMessage's Data Source Window to Define the Data Source for XML Message Elements (in Oracle Developer Forms)" shows how you can use *i*Message to define an XML message. This screenshot also illustrates the XML message elements and structure as well as the associated source SQL query.

A number of steps are involved when using *i*Message to define your XML message sets. These include the following:

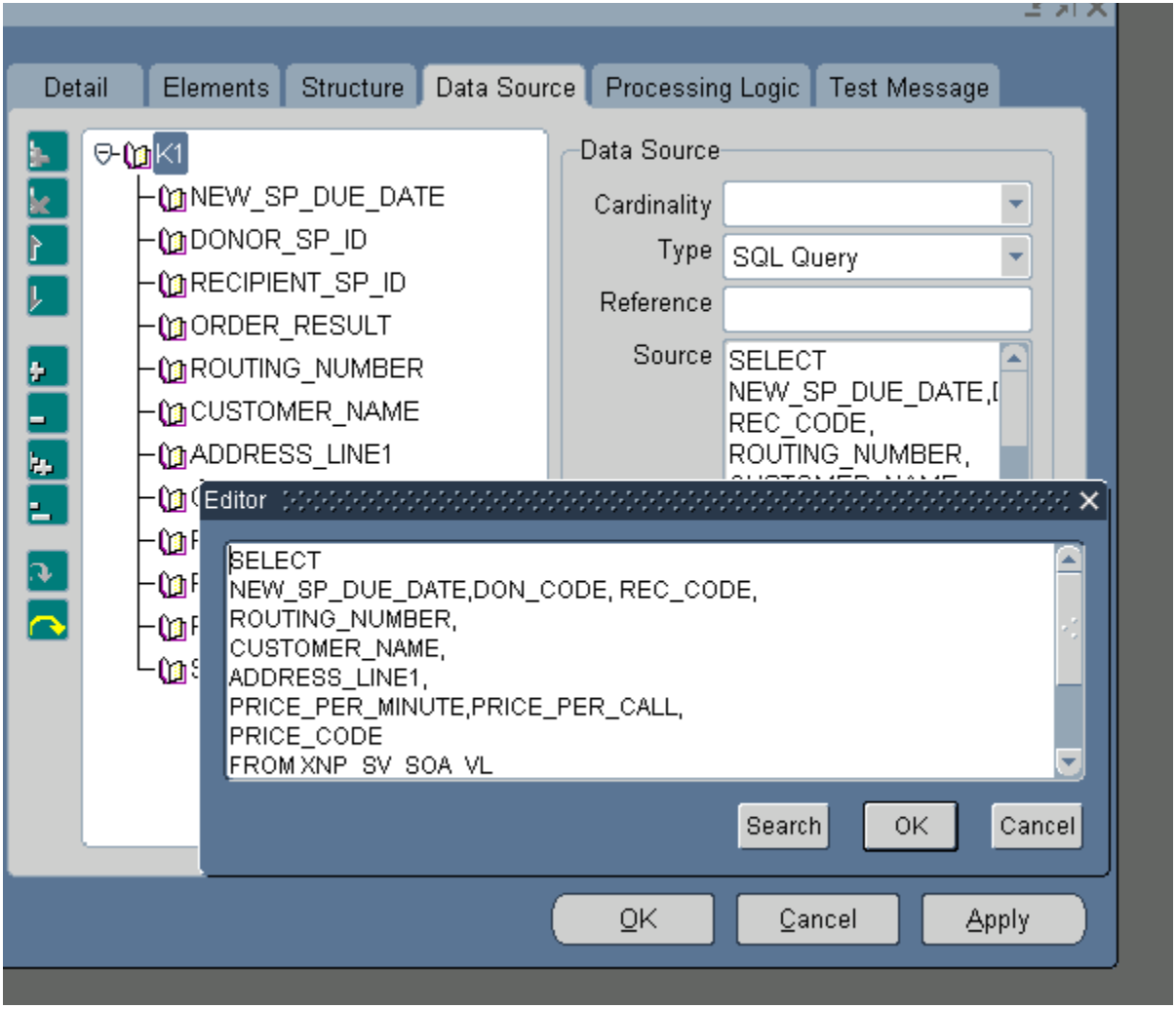
- *Defining Messages*. Messages can be defined by specifying all the elements (attributes) and their structural relationships. Other constraints like mandatory or optional, maximum data length and default values can also be specified.

- *Adding Message Details*. The Type Field. The Internet Message Studio can also be used to define application events. The key difference between messages and events are that messages are used for communication between application systems and events can be used to broadcast or multi-cast state changes in business objects. In addition, the studio also helps to define timer messages.

  Events defined using the Internet Message Studio are published to both external and internal application systems. "Internal applications" can register a PL/SQL callback procedure via the "event Publisher" screens or the above defined API and will get executed when an event is published. "External Applications" by definition do not register callback procedures but will have an adapter running to relay the published event to the remote system. External applications can register for an event using the default subscribers screen. A good example for internal applications is Oracle's SDP and Installed Base running on a single Oracle instance.

- *Description*. The description provides the context in which the message will be used.
- *Display Name*. The Display Name is the descriptive name of the message.
- *Adding Message Elements*
- *Building Message Structure*. The structure of the message defines the hierarchical relationship of the message elements. Only predefined elements can be part of this hierarchy. Please refer to the user guide for more information on building the message structure. The message structure can be viewed as an inverted tree, with the root as the top most element.
- *Root Element*. By default the Internet Message Studio includes 'MESSAGE' as a root element and the message being defined is the child of the root element. Please note that the root element is not visible and is implicitly defined by the Internet Message Studio. The root element should never be deleted. Elements not in the structure will not appear in the message.
- *Defining Data Source*. The next step in defining a message is to define the data source for message elements. Message elements can get their values from PL/SQL function calls and SQL queries. In addition, data can also be obtained as SDP Order parameters, SDP Work Item parameters or a Fulfillment Action parameters. See Table 12–1.

Figure 12–1 Using iMessage's Data Source Window to Define the Data Source for XML Message Elements (in Oracle Developer Forms)



Supported datatypes include

- PL/SQL Functions. A PL/SQL function can be executed at runtime to obtain the value of a message element. The specified PL/SQL function call can pass arguments by referring to any of the defined message elements. The PL/SQL function can also refer to any of the selected columns defined as a data source

on some upper level message elements. The function should be specified in the source field of the user interface and the return type should be same as the type specified for the message element.

- SQL Queries. A SQL query can be used to derive data for the message elements. The columns in the SQL query can be used as a reference for other message elements provided those message elements are defined at a lower level in the tree hierarchy.

Other data types are SDP Order Parameters, SADP Work Item Parameters, and SDP Fulfilment Action Parameters.

Part V

Developing Oracle XML Applications: A - Z

This section includes a detailed step by step explanation of how to build a B2B XML application from start to finish. This B2B XML application also illustrates how to present the same information to different devices.

Other chapters in Part V describe how to use JDeveloper and Internet File System (iFS) to build XML-based applications.

Part V contains the following chapters:

13

B2B XML Application: Step by Step

This chapter contains the following topics:

Introduction to the B2B XML Application

This chapter describes all the steps and scripts you need to build your demo B2B XML application.

These scripts are available for download from the Oracle Technology Network (OTN) site: http://technet.oracle.com/tech/xml.

Requirements for Running the B2B XML Application

The following lists requirements to build and run the B2B XML application:

- Client:
  - Operating system: Windows NT. The three .bat files used in this application are Windows specific. However you could rewrite these in shell script for UNIX systems.
  - Tools: JDeveloper 3.1 or higher: 208Mb
  - XML and XSL editors: Any editor. You can also use any text editor
  - Browser: Such as IE5.0, Netscape 5, or higher, and a PDA browser, such as HandWeb.
- Middle Tier:
  - Development environment needs 513Kb
  - Runtime environment only needs 135Kb
  - XSQL Servlet including the XML Parser for Java and XSU for Java
  - HTTP Listener
- Server:
  - Any Oracle and Java enabled server, such as Oracle8*i* Release 3 (8.1.7) or higher

Building the B2B XML Application: Overview

This XML application and demo illustrates a content management and B2B Messaging implementation. The main transactions in this application are as follows:

- A Retailer (R) places an order from any device, such as a browser, cell phone, or PDA (Personal Digital Assistant)

- The Supplier (S) is alerted that an order is received. After verifying inventory and the retailer's credit, the Supplier then clicks on the "Ship" button.
- Retailer and Supplier views the order's shipping status from any device

Problem

Retailers (R) need to automate the ordering of goods from several suppliers (Supplier (S)) and be able to place the order view the order status from any device.

Solution

This solution implements the following:

- Oracle XML components. To transform the HTML (or other format) order data received from the Retailer's web site into XML documents.
- Oracle8*i* Database(s). This solution assumes both the Retailer and Supplier are storing their data in Oracle8i databases.
- An AQ Broker -Transformer. This AQ application manages the flow of orders between the Retailer and Supplier. The Retailer submits the order in an AQ queue. The interested Supplier picks up (READS or dequeues) the order from the queue. AQ is also used to extract intelligence regarding the flow of orders. Each order is an XML message. This message is transformed into formats recognizable by both the Retailer and Supplier.

Tasks Identified

The main tasks are shown in Figure 13–2.

1. The Retailer enters an order from their Browser, Personal Digital Assistant (PDA), or cell phone.
2. When the Retailer validates his order, the order is transformed into XML using the XSQL Servlet.
3. The Retailer application sends the XML order to the AQ Broker.
4. AQ messaging is used to send the XML order data. The retailer views their order status as "Pending". AQ Broker reformats the XML order into a format understood by the Supplier.
5. The Supplier application inserts the order into the Supplier database.
6. The Supplier application parses the order and sends an alert to the Supplier that an order has been received and is waiting processing.

7. Once the Supplier hits "Shipped" on his screen, AQ messaging is used to return the XML order status data to the AQ Broker. The AQ Broker transforms the returned XML Order status into a format recognized by the Retailer.
8. The Supplier receives the reformatted XML order status message. The Retailer application updates the Retailer database with the new order status. The Retailer views the order status, which is now "Shipped".

The detailed tasks involved, screens viewed, and scripts used, are described in "Running the B2B XML Application: Detailed Procedure". and illustrated in Figure 13–2, "B2B XML Application: Main Components"

XML and Oracle Components Used

- XSQL Servlet. This includes the XML-SQL Utility (XSU), XML Parser for Java Version 2, and XSL-T Processor
- Oracle8*i*

Tools Used

JDeveloper

> **Note:** No pre-authored (static) XML documents are used in this B2B XML application. All XML documents in this application are dynamically generated from the database.

Why Transform Data to XML?

Retailers and Suppliers use different formats.

Because Retailers use different order form formats, the Retailer's order data is transformed into XML so that any Supplier can recognize and process their orders.

Suppliers use different formats for their order status and acknowledgement data. This data is converted to XML so that any Retailer can recognize the order status and acknowledgement.

> **Note:** This solution uses a finite set of two predetermined customer order document formats.

- *Retailer's Order Data*: The order data, stored in the Retailer Database R, is transformed by the AQ Broker using the appropriate XSL stylesheet into a format recognized by the specific Supplier.
- *Supplier's Order Status Data*: This data is transformed by the AQ Broker using the appropriate XSL stylesheet into a format recognized by the specific Retailer.

> **Note:** The Transformer API and associated tables can reside anywhere, including the Retailer's or Supplier database.

Figure 13–1 illustrates the overall flow of the Retailer-Supplier transaction. The Retailer enters the order.

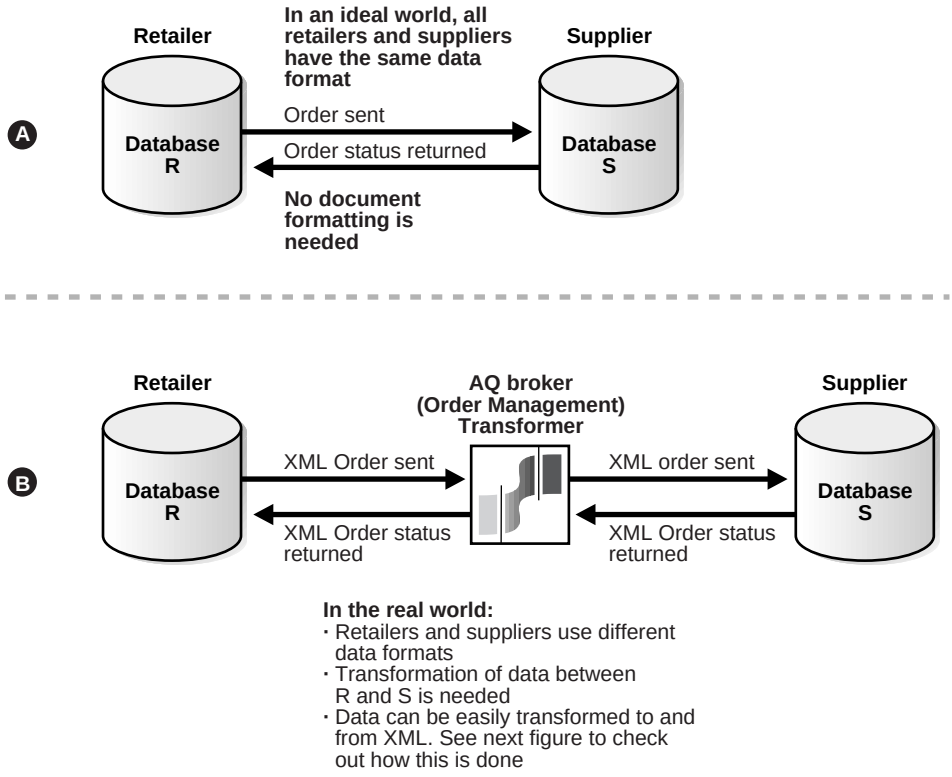
- In an ideal world, if the order document format of every Retailer and every Supplier were the same, the process would be simply as shown in A.
- In the real world, order document formats of each Retailer and Supplier typically differ. Making these transactions seamless is possible by converting the data to XML. By applying XSL stylesheets the data format can then customized and displayed by any device and in multiple formats.

Why Use Advanced Queueing (AQ)?

Using AQ in this application has the following advantages:

- AQ *manages the flow* of orders from Retailers to Suppliers and order status updates and acknowledgements from Suppliers to Retailers.
- AQ *separates the Retailer from Supplier* so that any Retailer can place their order in the same queue and any Supplier can simply pick up the orders from that same queue. In other words it facilitates a simple implementation of a many-to-many scenario.
- AQ can also *extract intelligence* about the orders being processed

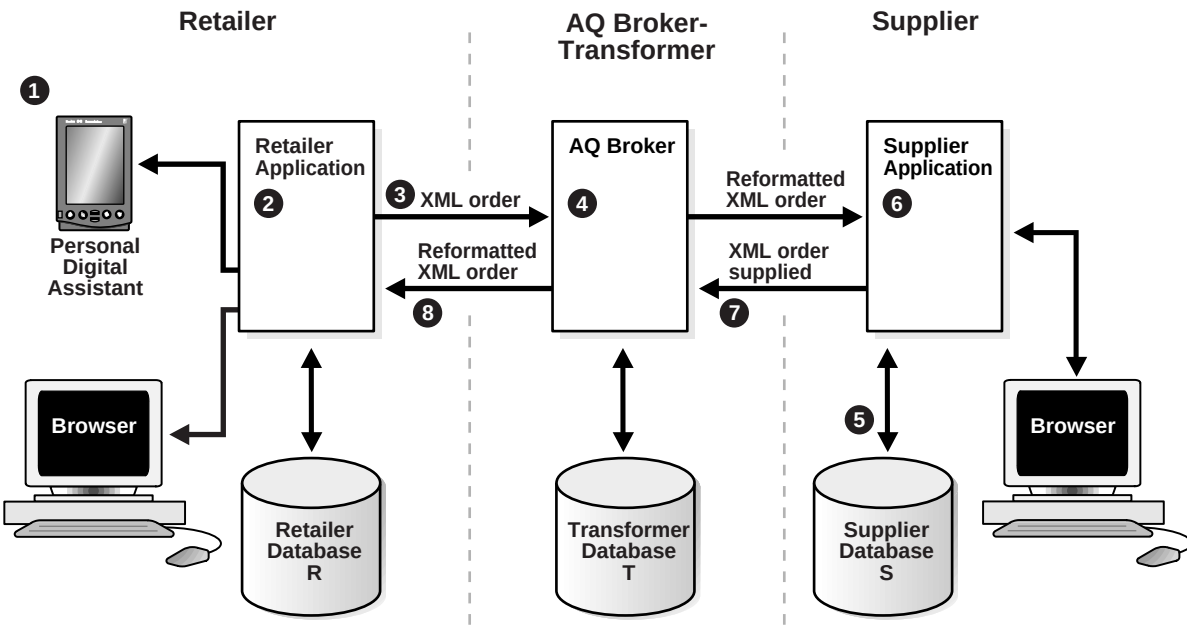
Figure 13–1 Why Transform Data to XML?: Retailer's Order Data Can be recognized by Any Supplier - Supplier's Order Status and Acknowledgement Can be Recognized by any Retailer



B2B XML Application: Main Components

Figure 13–2 shows the main components used in this B2B XML application. The Retailer orders good from a Supplier and receives a confirmation from the Supplier that the goods have been shipped. The detailed transaction diagram of the process is illustrated in Figure 13–5.

Figure 13–2 B2B XML Application: Main Components



Overview of Tasks to Run the B2B XML Application

The schemas used in the B2B XML application you are about to build, are illustrated in Figure 13–3.

To run the B2B XML application carry out the following tasks as described:

- 1 Set Up Your Environment to Run the B2B XML Application
- 2 Run the B2B Application
- 3 End the B2B Application Session

The details for running the B2B XML application including what you will see on your browser, are provided in "Running the B2B XML Application: Detailed Procedure" on page 13-36. You will also see typical screenshots of what the Retailer and Supplier see.

Figure 13–3 B2B XML Retailer (Customers) and Supplier Schema

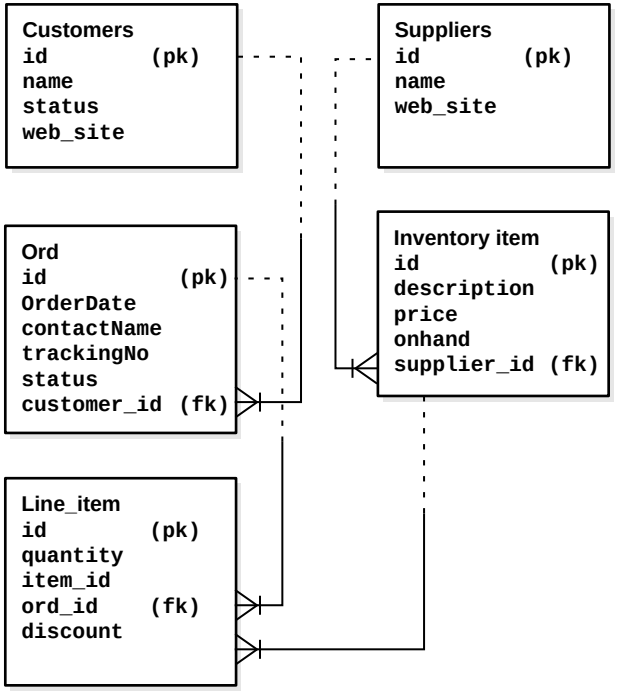
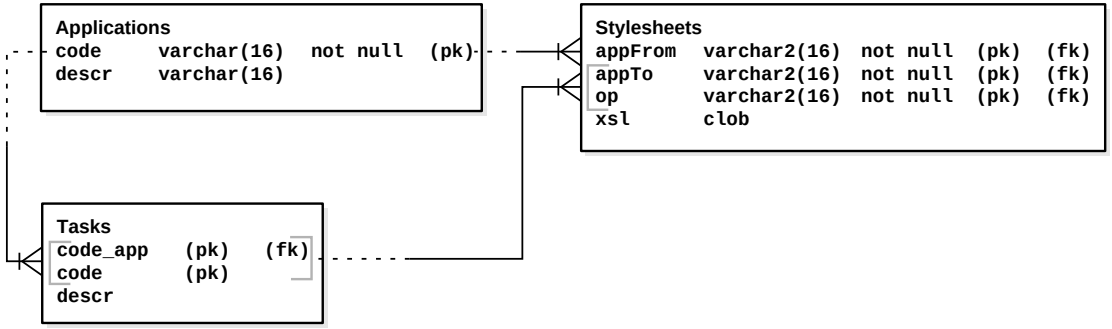


Figure 13–4 B2B XML AQ Broker Schema: Stylesheets



1 Set Up Your Environment to Run the B2B XML Application

1. Start your Apache or other Web Server.
2. Start your Browser, such as IE5
3. Log on
4. To set up all the schemas you will need to run the B2B XML application, follow these steps:

   Create the Retailer and Supplier schemas. See "B2B XML Application: Main Components"

   - Connect to the database however you like.
   - Run **buildAll.sql**. The script will ask you for your system password to create the requested users.
5. Create the AQ Schema
   - On a convenient machine, run the SQL script, **mkAQUser.sql**.
   - Connected as aqMessBrok / aqMessBrok, run the script, **mkQ.sql**
6. Create the XSL Tables
   - Still connected, run the script, **mkSSTables.sql**
   - Run **setup.sql** to install the XSL Stylesheets in the database.
   - Test it by running the GUIStylesheet java class, after changing the connections as described in the next step.
7. Modify the connections
   - Modify the JDBC Connection parameters in the following files:
     * AppCste.java
     * retail.bat
     * supplier.bat
     * PlaceOrder.xsql
   - Finally, modify XSQLConfig.xml to create a connection named retail on retailer / retailer.
   - Recompile all the files before going on.
8. Before running the B2B XML application, run the script named **reset.sql** to reset the AQ environment.

9. Modify and run the three bat files for the Broker, Suppler, and Retailer

   - Modify the .bat files: There are three mains used and these are launched from the following .bat files:

     * Broker.bat for the message broker
     * Supplier.bat for the supplier
     * Retail.bat for the retailer

     First modify the .bat files for your environment as follows:

     * *verbose*: If set to y or true, gives a lot of detail about the received messages.
     * *step*: If set to y or true, asks the user to hit return after each processing step. If step has a numeric value, it'll be considered, in milliseconds, as the time to wait between each step before going on

     `Retail.bat` and `Supplier.bat` also accept a -dbURL parameter, describing the URL used to get you connected to the database in question. The default URL is : jdbc:oracle:thin:@localhost:1521:ORCL.

2 Run the B2B Application

1. Run **broker.bat**, **supplier.bat**, and **retailer.bat**
2. Check the StyleSheet utility by running **GUIStylesheet.class**

These stylesheets are used by the Broker to process the documents it receives.

Details for running the B2B XML application including what you will see on your browser, are provided in "Running the B2B XML Application: Detailed Procedure".

3 End the B2B Application Session

1. To finish the B2B XML application

   Run the Java class, **stopAllQueues**, or the script named **stopQ.bat**

2. Stop Apache or your Web Server.

XML B2B Application: Setting Up the Database Schema

The following schema scripts are provided here in the order in which they should be executed:

- Create and Build the Retailer and Supplier Schemas
  - SQL Example 1: Set up the Retailer and Supplier Environment — BuildAll.sql
  - This calls, SQL Example 2: Create and Populate the Retailer-Supplier Schema — BuildSchema.sql
- Create the AQ Environment and Queue Tables
  - SQL Example 3: Set Up the Environment for AQ — mkAQUser.sql
  - SQL Example 4: Call the AQ Queue Creation Scripts — mkQ.sql. This calls:
    - SQL (PL/SQL) Example 5: Create Table, AppOne_QTab — mkQueueTableApp1.sql
    - SQL (PL/SQL) Example 6: Create Table, AppTwo_QTab — mkQueueTableApp2.sql
    - SQL (PL/SQL) Example 7: Create Table, AppThree_QTab — mkQueueTableApp3.sql
    - SQL (PL/SQL) Example 8: Create Table, AppFour_QTab — mkQueueTableApp4.sql
- Create the Broker Schema Including XSL Stylesheet Table
  - SQL Example 9: Create Broker Schema — mkSSTables.sql.

    This calls:
  - SQL (PL/SQL) Example 10: Input XSL data into CLOB. Populate the Broker Schema — setup.sql
- Cleaning Up Your Environment and Preparing to Rerun Application
  - SQL Example 11: Stops and Drops Queue Applications. Starts Queue Applications — reset.sql

SQL Calling Sequence

The following list provides the SQL example calling sequence. The .sql extension for each file has been omitted. The notation "<---" implies "calls", for example, BuildAllsql <----- BuildSchema implies that BuildAllsql calls BuildSchema.

- BuildAll.sql <---- BuildSchema.sql
- mkAQuser.sql
- mkQ.sql
  - <---- mkQueueTableApp1
  - <---- mkQueueTableApp2
  - <---- mkQueueTableApp3
  - <---- mkQueueTableApp4
- mkSSTables.sql <---- setup.sql
- reset.sql
  - <---- stopQueueApp1
  - <---- stopQueueApp2
  - <---- stopQueueApp3
  - <---- stopQueueApp4
  - <---- dropQueueApp1
  - <---- dropQueueApp2
  - <---- dropQueueApp3
  - <---- dropQueueApp4
  - <---- createQueueApp1
  - <---- createQueueApp2
  - <---- createQueueApp3
  - <---- createQueueApp4
  - <---- startQueueApp1
  - <---- startQueueApp2
  - <---- startQueueApp3

- <---- startQueueApp4

Create and Build the Retailer and Supplier Schemas

These schema scripts set up the Retailer and Supplier environment, users, tablespaces, quota, and so on. They also create and then populate the schemas.

- SQL Example 1: Set up the Retailer and Supplier Environment — BuildAll.sql. This calls:
- SQL Example 2: Create and Populate the Retailer-Supplier Schema — BuildSchema.sql

SQL Example 1: Set up the Retailer and Supplier Environment — BuildAll.sql

`BuildAll.sql` sets up the environment for the Retailer and Supplier schema. It calls `BuildSchema.sql` which creates the Retailer and Supplier schemas and then populates them with data.

```
--
-- buildall.sql builds all the schemas
--
accept sysPswd prompt 'Enter the system password
> ' hide
accept cStr    prompt 'Enter the connect string if any, including ''@'' sign (ie
@atp-1) > '
connect system/&sysPswd&cStr
drop user retailer cascade
/
drop user supplier cascade
/
col tablespace_name head "Available Tablespaces"
select tablespace_name from dba_tablespaces
/
prompt
accept userTbsp prompt 'What is the DEFAULT Tablespace name ?   > '
accept tempTbsp prompt 'What is the TEMPORARY Tablespace name ? > '
prompt

create user retailer identified by retailer
default tablespace &userTbsp
temporary tablespace &tempTbsp
quota unlimited on &userTbsp
/
grant connect, resource, create any directory to retailer
```

```
/
create user supplier identified by supplier
default tablespace &userTbsp
temporary tablespace &tempTbsp
quota unlimited on &userTbsp
/
grant connect, resource, create any directory to supplier
/
prompt Now populating Supplier, hit [Return]
pause
connect supplier/supplier&cStr
@buildSchema
prompt Now populating Retailer, hit [Return]
pause
connect retailer/retailer&cStr
@buildSchema
prompt done !
```

SQL Example 2: Create and Populate the Retailer-Supplier Schema — BuildSchema.sql

`BuildSchema.sql` is called from `BuildAll.sql`. It creates, populates, and builds the Retailer and Supplier schema.

This script creates and populates the following five tables:

- Customers
- Suppliers
- Inventory_item
- Ord
- Line_item

See Figure 13–3 for an illustration of this schema.

```
--
-- buildSchema.sql drops then creates all the tables for the B2B XML Application
--
drop trigger line_item_insert_trigger;
drop table line_item;
drop table ord;
drop table customer;
drop table inventory_item;
```

```
drop table supplier;
drop sequence ord_seq;
drop sequence customer_seq;
drop sequence line_item_seq;
drop sequence supplier_seq;
drop sequence inventory_item_seq;

prompt
prompt Creating sequences...
prompt
prompt
prompt Creating sequence ORD_SEQ
create sequence ord_seq start with 101;

prompt Creating sequence CUSTOMER_SEQ
create sequence customer_seq start with 201;

prompt Creating sequence LINE_ITEM_SEQ
create sequence line_item_seq start with 1001;

prompt Creating sequence SUPPLIER_SEQ
create sequence supplier_seq start with 301;

prompt Creating sequence INVENTORY_ITEM_SEQ
create sequence inventory_item_seq start with 401;

prompt
prompt
prompt Creating tables...
prompt
prompt
--
-- ***** Create table CUSTOMERS ******
--
prompt Creating table CUSTOMER
create table customer(
  id         number,
  name       varchar2(30),
  status     varchar2(8),
  web_site   varchar2(40),
  constraint
     customer_pk
     primary key (id)
);
--
```

```
-- ***** Create table SUPPLIERS ******
--
prompt Creating table SUPPLIER
create table supplier(
  id         number,
  name       varchar2(30),
  web_site   varchar2(40),
  constraint
     supplier_pk
     primary key (id)
);
--
-- ***** Create table INVENTORY_ITEM ******
--
prompt Creating table INVENTORY_ITEM
create table inventory_item(
  id             number,
  description    varchar2(30),
  price          number(8,2),
  onhand         number,
  supplier_id    number,
  constraint
     inventory_item_pk
     primary key (id),
  constraint
     supplied_by
     foreign key (supplier_id) references supplier
);
--
-- ***** Create table ORD ******
--
prompt Creating table ORD
create table ord (
  id             number,
  orderDate      date,
  contactName    varchar2(30),
  trackingNo     varchar2(20),
  status         varchar2(10),
  customer_id    number,
  constraint
     ord_pk
     primary key (id),
  constraint
     order_placed_by
     foreign key (customer_id) references customer
```

```
);

prompt Creating table LINE_ITEM
create table line_item(
  id             number,
  quantity       number,
  item_id        number,
  ord_id         number,
  discount       number,
  constraint
     line_item_pk
     primary key (id),
  constraint
     item_ordered_on
     foreign key (ord_id) references ord,
  constraint
     order_for_item
     foreign key (item_id) references inventory_item
);

prompt
prompt
prompt Inserting data...
prompt
prompt
prompt Inserting values into SUPPLIER and INVENTORY_ITEM
prompt

insert into supplier values( supplier_seq.nextval,'DELL','http://dell.com');
insert into inventory_item values( inventory_item_seq.nextval,'Optiplex GXPro',
1500, 27, supplier_seq.currval );
insert into inventory_item values( inventory_item_seq.nextval,'Inspiron 7000',
2500, 49, supplier_seq.currval );
insert into inventory_item values( inventory_item_seq.nextval,'PowerEdge 6300',
7500, 16, supplier_seq.currval );
insert into inventory_item values( inventory_item_seq.nextval,'Inspiron 3000',
2500, 0, supplier_seq.currval );
insert into inventory_item values( inventory_item_seq.nextval,'Inspiron 2000',
2500, 0, supplier_seq.currval );

insert into supplier values( supplier_seq.nextval, 'HP', 'http://hp.com');
insert into inventory_item values( inventory_item_seq.nextval, 'LaserJet 6MP',
899, 123, supplier_seq.currval );
insert into inventory_item values( inventory_item_seq.nextval, 'Jornada 2000',
450, 1198, supplier_seq.currval );
```

```
insert into inventory_item values( inventory_item_seq.nextval, 'HP 12C', 69,
801, supplier_seq.currval );
insert into inventory_item values( inventory_item_seq.nextval, 'LaserJet 2', 69,
3, supplier_seq.currval );


insert into inventory_item values( inventory_item_seq.nextval,'Jaz PCMCIA
adapter', 125, 54, supplier_seq.currval );
insert into inventory_item values( inventory_item_seq.nextval,'8860 Digital
phone', 499, 12, supplier_seq.currval );
insert into inventory_item values( inventory_item_seq.nextval,'Jaz carrying
bag', 20, 66, supplier_seq.currval );



insert into inventory_item values(supplier_seq.nextval,'Intel',
'http://www.intel.com');
prompt Inserting values into CUSTOMER
prompt


insert into ord values( ord_seq.nextval, sysdate, 'George', 'AX'||ord_
seq.currval, 'Pending', 201);
insert into line_item values (line_item_seq.nextval, 2, 410,ord_seq.currval, 0);
insert into line_item values (line_item_seq.nextval, 1, 402,ord_seq.currval, 0);
insert into line_item values (line_item_seq.nextval, 1, 406,ord_seq.currval, 0);

insert into ord values(ord_seq.nextval,sysdate,'Elaine','AX'||ord_seq.currval,
create trigger line_item_insert_trigger
  before insert on line_item for each row
  begin
     select line_item_seq.nextval into :new.id from dual ;
  end;
/

commit;
```

Create the AQ Environment and Queue Tables

Run the AQ schema scripts as follows:

- SQL Example 3: Set Up the Environment for AQ — mkAQUser.sql
- SQL Example 4: Call the AQ Queue Creation Scripts — mkQ.sql. This calls:
  - SQL (PL/SQL) Example 5: Create Table, AppOne_QTab — mkQueueTableApp1.sql
  - SQL (PL/SQL) Example 6: Create Table, AppTwo_QTab — mkQueueTableApp2.sql
  - SQL (PL/SQL) Example 7: Create Table, AppThree_QTab — mkQueueTableApp3.sql
  - SQL (PL/SQL) Example 8: Create Table, AppFour_QTab — mkQueueTableApp4.sql

SQL Example 3: Set Up the Environment for AQ — mkAQUser.sql

The following SQL script sets up the environment for using AQ, creates user aqMessBrok, creates default and temporary tablespace, grants execute privileges on the AQ PL/SQL packages dbms_aqadm and dbms_aq to aqMessBrok.

```
set ver off
set scan on
prompt Creating environment for Advanced Queuing
accept mgrPsw prompt 'Please enter the SYSTEM password
> ' hide
accept cStr   prompt 'Please enter the the DB Alias if any, WITH the @ sign (ie
@Ora8i)> '
connect system/&mgrPsw&cStr

col tablespace_name head "Available Tablespaces"
select tablespace_name from dba_tablespaces
/
Prompt
accept userTbsp prompt 'What is the DEFAULT Tablespace name ?   > '
accept tempTbsp prompt 'What is the TEMPORARY Tablespace name ? > '
prompt

prompt Creating aqMessBrok
create user aqMessBrok identified by aqMessBrok
default tablespace &userTbsp
temporary tablespace &tempTbsp
```

```
quota unlimited on &userTbsp
/
grant connect, resource, aq_administrator_role, create any directory to
aqMessBrok
/
grant execute on dbms_aqadm to aqMessBrok
/
grant execute on dbms_aq to aqMessBrok
/
```

SQL Example 4: Call the AQ Queue Creation Scripts — mkQ.sql

This script calls four scripts to create the AQ queue tables.

```
@mkQueueTableApp1
@mkQueueTableApp2
@mkQueueTableApp3
@mkQueueTableApp4
```

SQL (PL/SQL) Example 5: Create Table, AppOne_QTab — mkQueueTableApp1.sql

This script is called from mkQ.sql. It calls the dbms_aqadm.create_queue_table procedure to create queue table 1, AppOne_QTab.

```
execute dbms_aqadm.create_queue_table (queue_table => 'AppOne_QTab', queue_
payload_type => 'RAW');
```

SQL (PL/SQL) Example 6: Create Table, AppTwo_QTab — mkQueueTableApp2.sql

This script is called from mkQ.sql. It calls the dbms_aqadm.create_queue_table procedure to create queue table 2, AppTwo_QTab.

```
execute dbms_aqadm.create_queue_table (queue_table => 'AppTwo_QTab', queue_
payload_type => 'RAW');
```

SQL (PL/SQL) Example 7: Create Table, AppThree_QTab — mkQueueTableApp3.sql

This script is called from mkQ.sql. It calls the dbms_aqadm.create_queue_table procedure to create queue table 3, AppThree_QTab.

```
execute dbms_aqadm.create_queue_table (queue_table => 'AppThree_QTab', queue_
```

```
payload_type => 'RAW');
```

SQL (PL/SQL) Example 8: Create Table, AppFour_QTab — mkQueueTableApp4.sql

This script is called from mkQ.sql. It calls the dbms_aqadm.create_queue_table procedure to create queue table 4, AppFour_QTab.

```
execute dbms_aqadm.create_queue_table (queue_table => 'AppFour_QTab', queue_
payload_type => 'RAW');
```

Create the Broker Schema Including XSL Stylesheet Table

Run the following scripts to create and populate the stylesheets, tasks, and applications tables:

- SQL Example 9: Create Broker Schema — mkSSTables.sql.

  This calls:

- SQL (PL/SQL) Example 10: Input XSL data into CLOB. Populate the Broker Schema — setup.sql

SQL Example 9: Create Broker Schema — mkSSTables.sql

Run `mkSSTables.sql` to create the Broker schema. It creates and populates the following three tables:

- Stylesheets
- Tasks
- Applications

This schema is illustrated in Figure 13–4. This script then calls `setup.sql`.

```
prompt Building Stylesheets management tables.
prompt Must be connected as aqMessBrok (like the borker)
accept cStr prompt 'ConnectString (WITH @ sign, like @Ora8i) > '
connect aqMessBrok/aqMessBrok&cStr

drop table styleSheets
/
drop table tasks
/

drop table applications
/
create table applications
(
  code  varchar2(16) not null,
  descr varchar2(256)
)
/
alter table applications
  add constraint PK_APP
  primary key (code)
/
create table tasks
```

```
(
  code_app varchar2(16) not null,
  code     varchar2(16) not null,
  descr    varchar2(256)
)
/
alter table tasks
  add constraint PK_TASKS
  primary key (code_app,code)
/
alter table tasks
  add constraint TASK_FK_APP
  foreign key (code_app)
  references applications(code) on delete cascade
/
create table styleSheets
(
  appFrom varchar2(16) not null,
  appTo   varchar2(16) not null,
  op      varchar2(16) not null,
  xsl     clob
)
/
alter table styleSheets
  add constraint PK_SS
  primary key (appFrom,appTo,op)
/
alter table styleSheets
  add constraint SS_FK_FROM
  foreign key (appFrom)
  references applications(code)
/
alter table styleSheets
  add constraints SS_FK_TASK
  foreign key (appTo,op)
  references tasks(code_app,code)
/
@setup
```

SQL (PL/SQL) Example 10: Input XSL data into CLOB. Populate the Broker Schema — setup.sql

`setup.sql` installs stylesheet data into the XSL column (CLOB) of the stylesheets table. This script creates a procedure, `loadlob`. The script also uses PL/SQL packages `dbms_lob` and `dbms_output`.

```
prompt Installing the stylesheets
-- accept cStr prompt 'ConnectString (WITH @ sign, like @Ora8i) > '
-- connect aqMessBrok/aqMessBrok&cStr
prompt Creating LoadLob procedure
create or replace procedure loadLob (imgDir in varchar2,
                                     fname in varchar2,
                                     app_From in varchar2,
                                     app_To in varchar2,
                                     oper in varchar2) as
  tempClob  CLOB;
  fileOnOS  BFILE := bfilename(imgDir, fname);
  ignore    INTEGER;
begin
  dbms_lob.fileopen(fileOnOS, dbms_lob.file_readonly);
  select xsl
  into tempClob
  from StyleSheets S
  where s.APPFROM = app_From and
        s.APPTO = app_To and
        s.OP = oper
  for UPDATE;
dbms_output.put_line('External file size is: ' || dbms_lob.getlength(fileOnOS));
dbms_lob.loadfromfile(tempClob, fileOnOS, dbms_lob.getlength(fileOnOS));
dbms_lob.fileclose(fileOnOS);
dbms_output.put_line('Internal CLOB size is: '|| dbms_lob.getlength(tempClob));
exception
  When Others then
    dbms_output.put_line('Oooops : ' || SQLERRM);
end LoadLob;
/
show errors
set scan off

create or replace directory "LOB_DIR" as 'D:\xml817\references\olivier_new'
/
insert into applications values ('RETAIL', 'Origin')
/
insert into applications values ('SUPPLY', 'Destination')
```

```
/
insert into tasks values ('SUPPLY', 'NEW ORDER', 'Insert a new Order')
/
insert into tasks values ('RETAIL', 'UPDATE ORDER', 'Update an Order Status')
/

set serveroutput on
begin
 insert into StyleSheets values ('RETAIL','SUPPLY','NEW ORDER',EMPTY_CLOB());
 loadLob('LOB_DIR', 'one.xsl', 'RETAIL','SUPPLY','NEW ORDER');
 insert into StyleSheets values ('SUPPLY','RETAIL','UPDATE ORDER',EMPTY_CLOB());
 loadLob('LOB_DIR', 'two.xsl', 'SUPPLY','RETAIL','UPDATE ORDER');
exception
  when others then
  dbms_output.put_line('Error Occurred : ' || chr(10) || SQLERRM);
end;
/
commit
/
```

Cleaning Up Your Environment and Preparing to Rerun Application

Run `reset.sql` to clean up your environment and rerun this application.

- SQL Example 11: Stops and Drops Queue Applications. Starts Queue Applications — reset.sql

  This calls the following 16 PL/SQL scripts:

  - stopQueueApp1.sql
  - stopQueueApp2.sql
  - stopQueueApp3.sql
  - stopQueueApp4.sql
  - dropQueueApp1.sql
  - dropQueueApp2.sql
  - dropQueueApp3.sql
  - dropQueueApp4.sql
  - createQueueApp1.sql
  - createQueueApp2.sql

- createQueueApp3.sql
- createQueueApp4.sql
- startQueueApp1.sql
- startQueueApp2.sql
- startQueueApp3.sql
- startQueueApp4.sql

SQL Example 11: Stops and Drops Queue Applications. Starts Queue Applications — reset.sql

`reset.sql` script first stops all four queue applications by calling the stopQueueApp1 through 4, then drops them by calling dropQueueApp1 through 4, and restarts them by calling startQueueApp1 through 4.

The script also prompts you to Hit Return to Exit.

```
connect aqMessBrok/aqMessBrok
start stopQueueApp1
start stopQueueApp2
start stopQueueApp3
start stopQueueApp4
start dropQueueApp1
start dropQueueApp2
start dropQueueApp3
start dropQueueApp4
start createQueueApp1
start createQueueApp2
start createQueueApp3
start createQueueApp4
start startQueueApp1
start startQueueApp2
start startQueueApp3
start startQueueApp4
prompt Press [Return] to exit !
pause
exit
```

Stop Queue SQL Scripts

These four scripts are called from `reset.sql`. They use PL/SQL procedure `dbms_aqadm.stop_queue` to stop the queues.

stopQueueApp1.sql

```
execute dbms_aqadm.stop_queue(queue_name=>'AppOneMsgQueue');
```

stopQueueApp2.sql

```
execute dbms_aqadm.stop_queue(queue_name=>'AppTwoMsgQueue');
```

stopQueueApp3.sql

```
execute dbms_aqadm.stop_queue(queue_name=>'AppThreeMsgQueue');
```

stopQueueApp4.sql

```
execute dbms_aqadm.stop_queue(queue_name=>'AppFourMsgQueue');
```

Drop Queue SQL Scripts

These four scripts are called from `reset.sql`. They use PL/SQL procedure `dbms_aqadm.drop_queue` to drop the queues.

dropQueueApp1.sql

```
execute dbms_aqadm.drop_queue (queue_name=>'AppOneMsgQueue');
```

dropQueueApp2.sql

```
execute dbms_aqadm.drop_queue (queue_name=>'AppTwoMsgQueue');
```

dropQueueApp3.sql

```
execute dbms_aqadm.drop_queue (queue_name=>'AppThreeMsgQueue');
```

dropQueueApp4.sql

```
execute dbms_aqadm.drop_queue (queue_name=>'AppFourMsgQueue');
```

Create Queue SQL Scripts

These four scripts are called from `reset.sql`. They use PL/SQL procedure, `dbms_aqadm.create_queue` to create the queues.

createQueueApp1.sql

```
execute dbms_aqadm.create_queue (queue_name=>'AppOneMsgQueue', queue_
table=>'AppOne_QTab');
```

createQueueApp2.sql

```
execute dbms_aqadm.create_queue (queue_name=>'AppTwoMsgQueue', queue_
table=>'AppTwo_QTab');
```

createQueueApp3.sql

```
execute dbms_aqadm.create_queue (queue_name=>'AppThreeMsgQueue', queue_
table=>'AppThree_QTab');
```

createQueueApp4.sql

```
execute dbms_aqadm.create_queue (queue_name=>'AppFourMsgQueue', queue_
table=>'AppFour_QTab');
```

Start Queue SQL Scripts

These four scripts are called from `reset.sql`. They use PL/SQL procedure, `dbms_aqadm.start_queue` to start the queues.

startQueueApp1.sql

```
execute dbms_aqadm.start_queue(queue_name=>'AppOneMsgQueue');
```

startQueueApp2.sql

```
execute dbms_aqadm.start_queue (queue_name=>'AppTwoMsgQueue');
```

startQueueApp3.sql

```
execute dbms_aqadm.start_queue (queue_name=>'AppThreeMsgQueue');
```

startQueueApp4.sql

```
execute dbms_aqadm.start_queue (queue_name=>'AppFourMsgQueue');
```

dropOrder.sql

This SQL script deletes orders from the Retailer-Supplier database Customers table according to the customer's ID.

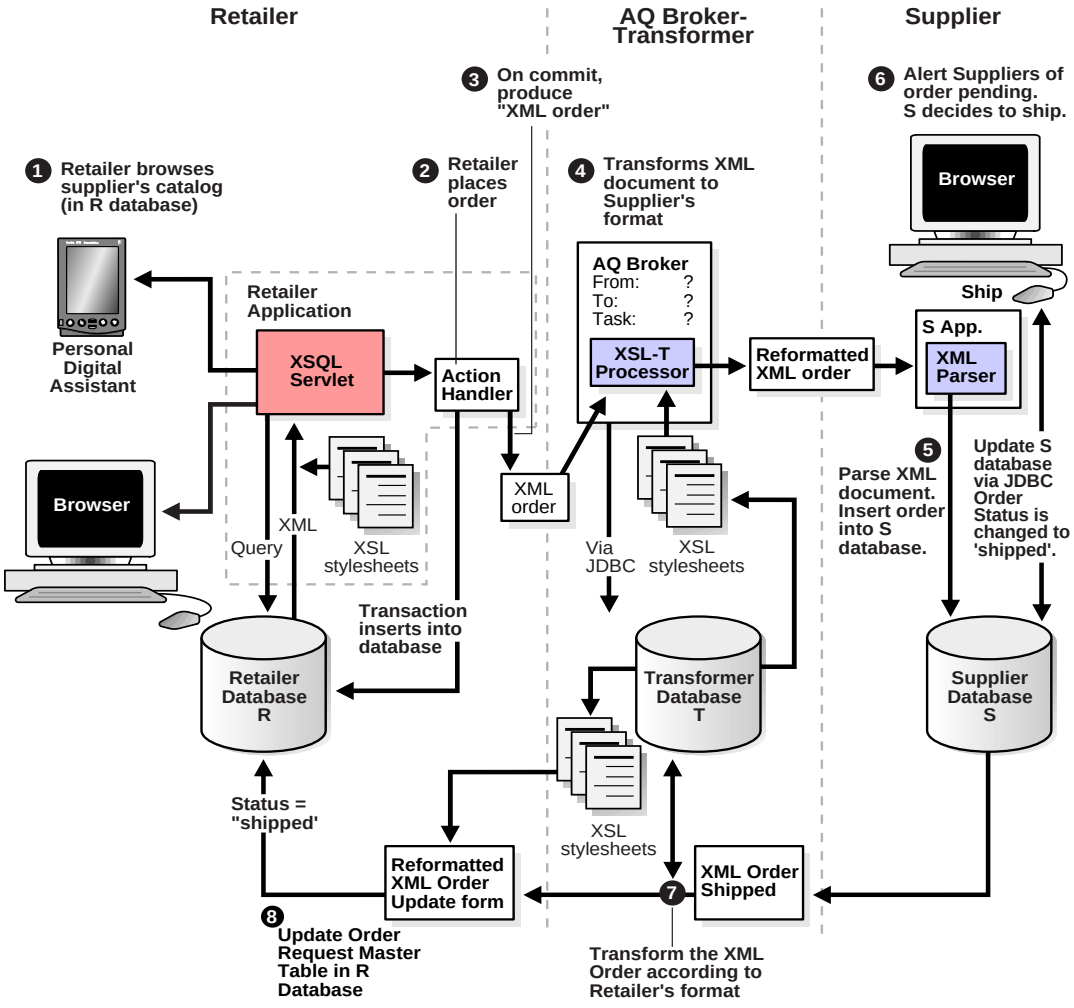
```
set ver off
accept CustName prompt 'Drop all for customer named > '

Delete LINE_ITEM I
```

```
Where I.ORD_ID in
(Select O.ID
 From ORD O
 Where O.CUSTOMER_ID in
 (Select C.ID
  From CUSTOMER C
  Where Upper(C.NAME) = Upper('&CustName')))
/
Delete ORD O
Where O.CUSTOMER_ID in
(Select C.ID
 From CUSTOMER C
 Where Upper(C.NAME) = Upper('&CustName'))
/
```

B2B XML Application: Data Exchange Flow

Figure 13–5 shows the detailed transaction diagram of the process when the Retailer orders good from a Supplier and receives a confirmation from the Supplier that the goods have been shipped.

Figure 13–5 Inter-Business Data Exchange: Using XML and AQ to send Retailer's Order to a Supplier and Receive Order Status and Acknowledgement from the Supplier



Retailer-Supplier Transactions

Figure 13–5 shows the business flow of the Retailer - Supplier transactions. These transactions are summarized here.

- 1 Retailer Browses the Supplier's OnLine "Hi-Tech Mall" Catalog
- 2 Retailer Places Order
- 3 Retailer Confirms and Commits to Sending the Order
- 4 AQ Broker-Transformer Transforms the XML Document According to the Supplier's Format
- 5 Supplier Application Parses Incoming Reformatted XML Order Document. Inserts Order into the Supplier Database
- 6 Supplier Application Alerts Supplier of Pending Order
- 7 AQ Broker-Transformer Transforms the XML Order According to Retailer's Format
- 8 Retailer Application Updates the Ord and Line_Item Tables

The detailed transactions and how to run the B2B XML application is provided in "Running the B2B XML Application: Detailed Procedure" on page 13-36.

1 Retailer Browses the Supplier's OnLine "Hi-Tech Mall" Catalog

The following Retailer transactions occur:

1. The Retailer logs in from their web site using XSL Servlet.
2. Retailer browses the Supplier's on-line catalog. Retailer selects a product and quantity.

2 Retailer Places Order

When the Retailer places the order, the Retailer then needs to either confirm the order and cost, by clicking on "Place Order", or cancel "Give Up" the order.

3 Retailer Confirms and Commits to Sending the Order

If Retailer confirms the order by clicking on, "Place Order", this triggers the generation of an XML document containing the order data. The Retailer application sends this XML order document to the Supplier by way of the AQ Broker-Transformer application.

The Action Handler "XSQL Script Example 5: Starts B2B Process — placeorder.xsql" of the XSQL Servlet is the key component in the whole process. It ensure that this transaction is inserted into the retailer database table, Ord.

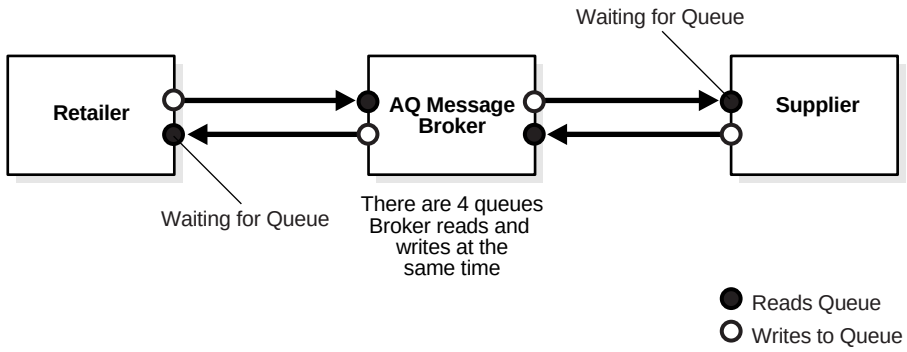
The Action Handler also sends the XML order on to the AQ Broker-Transformer.

4 AQ Broker-Transformer Transforms the XML Document According to the Supplier's Format

When the AQ Broker-Transformer receives the XML document the following actions transpire:

1. The AQ Broker-Transfomer waits for the queue [READS] from the Retailer that they have sent an order. See Figure 13–6.

Figure 13–6 B2B XML Application: AQ Messaging Flow



2. The AQ Broker receives the XML document order message, and determines the following information from the message:
   - FROM: From where the message is coming (from which Retailer)
   - TO: To where the message is going (to which Supplier)
   - OPERATION or TASK: What operation is needed to process this message
3. The AQ Broker-Transformer refers to the Stylesheets table and according to the From, To, and Task criteria, selects the appropriate XSL stylesheet. The stylesheets are stored in CLOBs in the Stylesheets table in the XSL column. AQ Broker-Transformer accesses the database and stylesheets by means of JDBC.

4. XSL-T Processor is informed by AQ Broker-Transformer application to apply the selected and retrieved XSL stylesheet to the XML document containing the order data. The XSL-T Processor outputs the reformatted XML order.
5. AQ Broker-Transformer uses AQ to send [WRITE] the transformed XML document to the "TO" Supplier destination.

> **Note:** If a DTD (XML Schema) is used, it would be applied before processing in the AQ Broker phase. In this example, for simplicity, we assume that the document is always sent in the same format.

The schema used by the AQ Broker-Transformer is shown in Figure 13–4.

5 Supplier Application Parses Incoming Reformatted XML Order Document. Inserts Order into the Supplier Database

When the Supplier receives the reformatted XML order document from the AQ Broker-Transformer, the following protocols transpire:

1. The Supplier waits for the queue from the AQ Broker-Transformer that a order is pending from a Retailer. The Supplier dequeues the AQ message.
2. The Supplier parses the XML document and INSERTs the order into the Supplier database by means of JDBC.

6 Supplier Application Alerts Supplier of Pending Order

When the Supplier application has inserted the XML document into the Supplier database the following actions transpire:

1. Supplier Application Alerts the Supplier of the Order. The order status is kept at "pending".
2. The Supplier, after checking if the product(s) ordered are available, and the Retailer's credit, decides to ship the product(s). Supplier clicks on "Ship".
3. The Supplier application updates the Supplier database Ord table's status column to "shipped".

7 AQ Broker-Transformer Transforms the XML Order According to Retailer's Format

1. AQ Broker-Transformer waits [READS] for a queue from the Supplier.

2. When the XML Order Shipped document is received, the AQ Broker-Transformer refers to the Stylesheets table in the Transformer database, and according to the From, To, and Task criteria, selects the appropriate XSL stylesheet. The stylesheets are stored in CLOBs in the Stylesheets table in the XSL column. AQ Broker-Transformer accesses the database and stylesheets by means of JDBC.
3. AQ Broker-Transformer uses AQ to send [WRITE] the reformatted XML order update document to the "TO" Retailer destination.

8 Retailer Application Updates the Ord and Line_Item Tables

1. Retailer application updates the Retailer database with new "shipped" order status information. The Ord table is updated.
2. This information is viewed by the Retailer from any device. The status is seen as "Shipped".

Running the B2B XML Application: Detailed Procedure

Figure 13–5 shows the detailed transaction and flow of the B2B XML application. The XML order document is sent from the Retailer through the AQ Broker-Transformer, to the Supplier and back to the Retailer.

Before running the B2B XML application, ensure that you have run the schema creation scripts described in "Overview of Tasks to Run the B2B XML Application".

The following steps explain the process and how to run this application.

- 1 Retailer Browses the Supplier's OnLine "Hi-Tech Mall" Catalog
- 2 Retailer Places Order
- 3 "Validate" Commits the Transaction. Retailer Application Produces the XML Order
- 4 AQ Broker-Transformer Transforms XML Document According to Supplier's Format
- 5 Supplier Application Parses the XML Document and Inserts the Order into the Supplier Database
- 6a Supplier Application Alerts Supplier of Pending Order
- 7 AQ Broker-Transformer Transforms XML Order into Retailer's Format
- 8 Retailer Application Updates the Ord Table and Displays the New Order Status to Retailer

1 Retailer Browses the Supplier's OnLine "Hi-Tech Mall" Catalog

See Figure 13–5 for the detailed procedural flow of the B2B XML application.

Note: We assume here that a copy of the Supplier's catalog is in the Retailer's database.

---

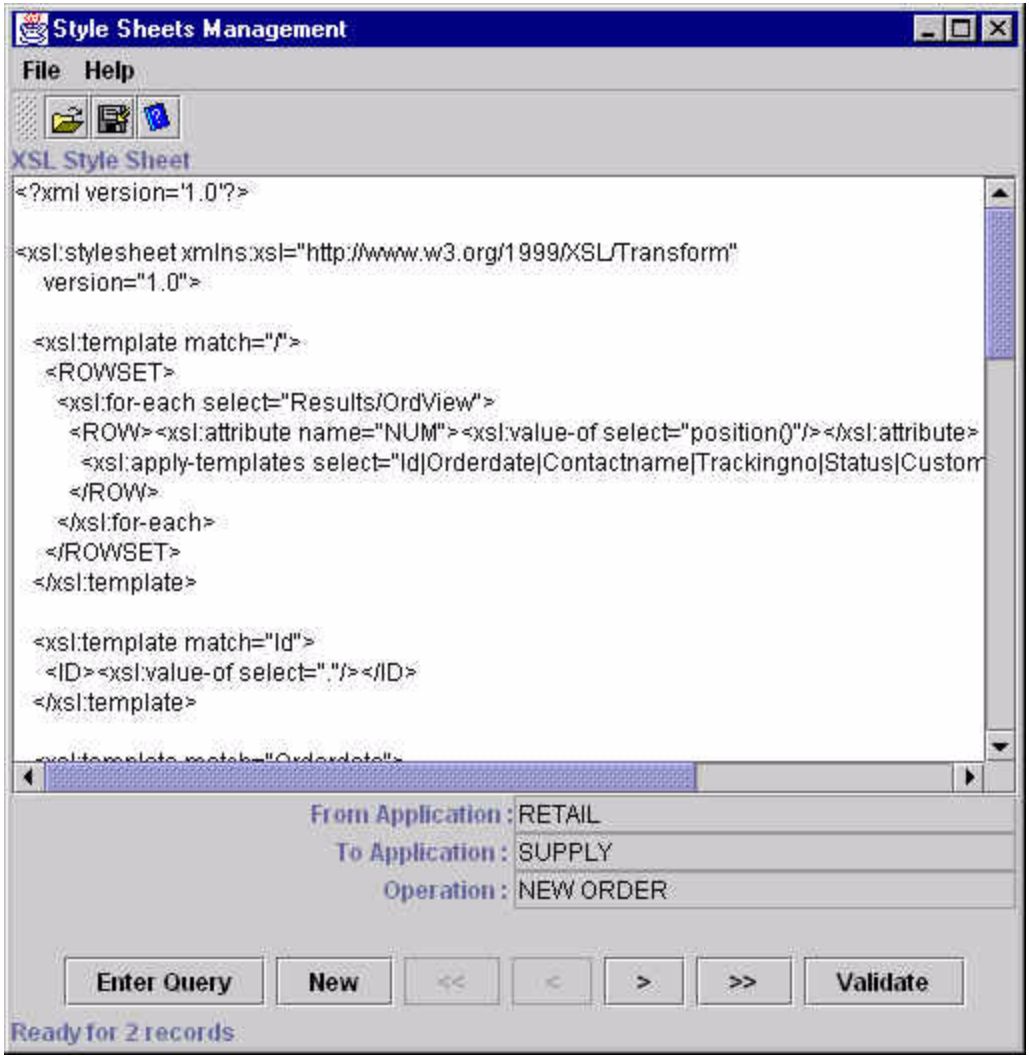
1. Check the StyleSheet utility to ensure it works by invoking `SS.bat`.

   Stylesheet Batch File: SS.bat

```
@echo off
@echo Stylesheet Util
D:\jdev31\java\bin\java -mx50m -classpath "D:\xml817\references\olivier_new;
D:\jdev31\lib\jdev-rt.zip;
D:\jdev31\jdbc\lib\oracle8.1.6\classes111.zip;
D:\jdev31\lib\connectionmanager.zip;
D:\jdev31\lib;
D:\jdev31\lib\oraclexsql.jar;
D:\jdev31\lib\oraclexmlsql.jar;
D:\jdev31\lib\xmlparserv2_2027.jar;
D:\jdev31\jfc\lib\swingall.jar;
D:\jdev31\jswdk-1.0.1\lib\servlet.jar;
D:\Ora8i\rdbms\jlib\aqapi11.jar;
D:\Ora8i\rdbms\jlib\aqapi.jar;
D:\XMLWorkshop\xmlcomp.jar;
D:\jdev31\java\lib\classes.zip"  B2BDemo.StyleSheetUtil.GUIStylesheet
```

   Using this utility you can browse the actual table, `Stylesheets`, in which the stylesheets are stored. These stylesheets are used by the AQ Broker-Transformer to process the documents it received. See Figure 13–7.

Figure 13–7 Checking the StyleSheet Utility



2. Start the Retailer application by running retailer.bat. See Figure 13–8.

Figure 13–8 Starting the Retailer Application

3. Start the AQ Broker-Transformer application by running broker.bat. See Figure 13–9.

Figure 13–9 Starting the AQ Broker-Transformer Application

4. Start the Supplier application by running supplier.bat. See Figure 13–10.

Figure 13–10 Starting the Supplier Application: "Supplier Watcher"

The three batch files for the Retailer, AQ Broker-Transformer (Broker), and Supplier applications are listed here:

retailer.bat

```
@echo off
@echo Retail Side
D:\jdev31\java\bin\java -mx50m -classpath
"D:\xml817\references\Ora817DevGuide;
D:\jdev31\lib\jdev-rt.zip;
D:\jdev31\jdbc\lib\oracle8.1.6\classes111.zip;
D:\jdev31\lib\connectionmanager.zip;
D:\jdev31\lib;
D:\jdev31\lib\oraclexsql.jar;
D:\jdev31\lib\oraclexmlsql.jar;
D:\jdev31\lib\xmlparserv2_2027.jar;
D:\jdev31\jfc\lib\swingall.jar;
D:\jdev31\jswdk-1.0.1\lib\servlet.jar;
D:\Ora8i\rdbms\jlib\aqapi11.jar;
D:\Ora8i\rdbms\jlib\aqapi.jar;
D:\XMLWorkshop\xmlcomp.jar;
D:\jdev31\java\lib\classes.zip"  B2BDemo.Retailer.UpdateMaster -step=1000
-verbose=y -dbURL=jdbc:oracle:thin:@atp-1.us.oracle.com:1521:ORCL
```

broker.bat

```
@echo off
@echo Broker
D:\jdev31\java\bin\java -mx50m -classpath
```

```
"D:\xml817\references\Ora817DevGuide;
D:\jdev31\lib\jdev-rt.zip;
D:\jdev31\jdbc\lib\oracle8.1.6\classes111.zip;
D:\jdev31\lib\connectionmanager.zip;
D:\jdev31\lib;D:\jdev31\lib\oraclexsql.jar;
D:\jdev31\lib\oraclexmlsql.jar;
D:\jdev31\lib\xmlparserv2_2027.jar;
D:\jdev31\jfc\lib\swingall.jar;
D:\jdev31\jswdk-1.0.1\lib\servlet.jar;
D:\Ora8i\rdbms\jlib\aqapi11.jar;
D:\Ora8i\rdbms\jlib\aqapi.jar;
D:\XMLWorkshop\xmlcomp.jar;
D:\jdev31\java\lib\classes.zip"  B2BDemo.Broker.MessageBroker -step=1000
-verbose=y
```

supplier.bat

```
@echo off
@echo Supplier
D:\jdev31\java\bin\java -mx50m -classpath
"D:\xml817\references\Ora817DevGuide;
D:\jdev31\lib\jdev-rt.zip;
D:\jdev31\jdbc\lib\oracle8.1.6\classes111.zip;
D:\jdev31\lib\connectionmanager.zip;
D:\jdev31\lib;D:\jdev31\lib\oraclexsql.jar;
D:\jdev31\lib\oraclexmlsql.jar;
D:\jdev31\lib\xmlparserv2_2027.jar;
D:\jdev31\jfc\lib\swingall.jar;
D:\jdev31\jswdk-1.0.1\lib\servlet.jar;
D:\Ora8i\rdbms\jlib\aqapi11.jar;
D:\Ora8i\rdbms\jlib\aqapi.jar;
D:\XMLWorkshop\xmlcomp.jar;
D:\jdev31\java\lib\classes.zip"  B2BDemo.Supplier.SupplierWatcher -step=1000
-verbose=y -dbURL=jdbc:oracle:thin:@atp-1.us.oracle.com:1521:ORCL
```

5. Finally, start the Client, from a browser, a PDA such as Palm Pilot, cell phone, or other device.
6. [Retailer] Log in. You will see a Welcome! screen. See Figure 13–11.

XSQL Script Example 2: Checking the ID of Users Logging In: getlogged.xsql

```
<?xml version="1.0"?>

<!--
```

```
 | Second script to be called.
 | Check if the user in known in the database.
 | $Author: olediour@us $
 | $Revision: 1.1 $
 +-->
<?xml-stylesheet type="text/xsl" media="HandHTTP"     href="PP.xsl"?>
<?xml-stylesheet type="text/xsl" media="Mozilla"      href="HTML.xsl"?>

<loginResult xmlns:xsql="urn:oracle-xsql"
             connection="retail"
             custName="XXX">
  <pageTitle>Hi-Tech Mall</pageTitle>
  <xsql:query tag-case="upper">
    <![CDATA[
    select C.ID, C.NAME
    from CUSTOMER C
    where Upper(C.NAME) = Upper('{@custName}')
    ]]>
    <xsql:no-rows-query>
      Select '{@custName}' as "unknown" from dual
    </xsql:no-rows-query>
  </xsql:query>
  <nextStep>inventory.xsql</nextStep>
  <returnHome>index.xsql</returnHome>

</loginResult>
```

This XSQL script calls the following XSL scripts:

- pp.xsl. See "XSL Stylesheet Example 1: Converts Results to HTML — html.xsl"
- html.xsl. See "XSL Stylesheet Example 2: Converts Results for Palm Pilot Browser — pp.xsl"

Figure 13–11 [Retailer]: Logging in from a Browser or PDA

7. [Retailer]: Click on 'Please Enter the Mall'.

XSQL Script Example 1: Displays First Hi-Tech Mall Screen — index.xsql

```
<?xml version="1.0"?>

<!--
 | This is the entry point in the application.
 | Notice that this script does not access the database.
 | $Author: olediour@us $
 | $Revision: 1.1 $
 +-->
<?xml-stylesheet type="text/xsl" media="HandHTTP"     href="PP.xsl"?>
<?xml-stylesheet type="text/xsl" media="Mozilla"      href="HTML.xsl"?>

<index xmlns:xsql="urn:oracle-xsql">
  <pageTitle>Hi-Tech Mall</pageTitle>
  <form action="getLogged.xsql" method="post">
    <field type="text" name="custName" prompt="Your ID"/>
    <button type="submit" label="Log In"/>
  </form>
</index>
```

8. [Retailer]: The resulting screen displays the Hi-Tech Mall Catalog product listing. Click on the product you are interested in. See Figure 13–12.

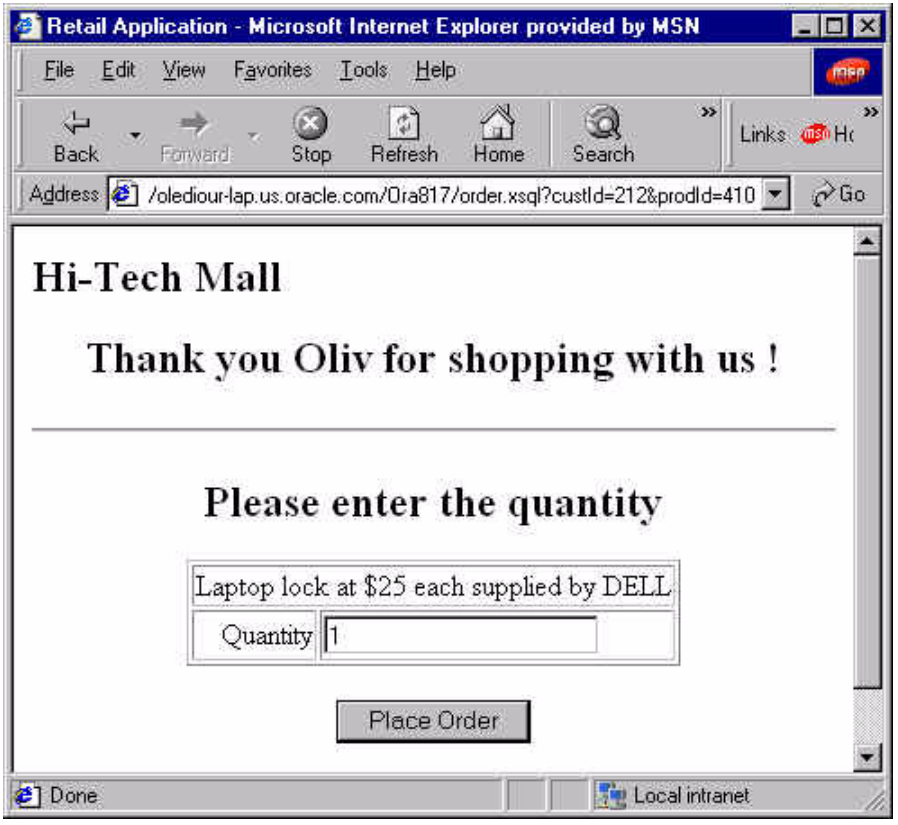
XSQL Script Example 3: Lists Catalog Products — inventory.xsql

```
<?xml version="1.0"?>

<!--
 | This is the third script called.
 | It produces the catalog from the Retailer's database.
 |
 | $Author: olediour@us $
 | $Revision: 1.1 $
 +-->
<?xml-stylesheet type="text/xsl" media="HandHTTP"     href="PP.xsl"?>
<?xml-stylesheet type="text/xsl" media="Mozilla"      href="HTML.xsl"?>

<inventory xmlns:xsql="urn:oracle-xsql"
           connection="retail"
           custId="000">
  <pageTitle>Hi-Tech Mall</pageTitle>
  <form action="order.xsql" method="post">
    <hiddenFields>
      <xsql:include-param name="custId"/>
    </hiddenFields>
    <theMart>
      <xsql:query tag-case="upper">
        <![CDATA[
        select I.ID,
               I.DESCRIPTION,
               I.PRICE,
               S.NAME
        from INVENTORY_ITEM I,
             SUPPLIER S
        where I.SUPPLIER_ID = S.ID
        ]]>
        <xsql:no-rows-query>
          Select 'No items !' as "Wow" from dual
        </xsql:no-rows-query>
      </xsql:query>
    </theMart>
  </form>
  <returnHome>index.xsql</returnHome>

</inventory>
```

Figure 13–12 [Retailer] Enter the Hi-Tech Mall (Mart) Catalog

9. [Retailer]: Enter the quantity you need and click the "Place Order" button. See Figure 13–13.

Figure 13–13 [Retailer]: Enter the Quantity and Click on "Place Order"



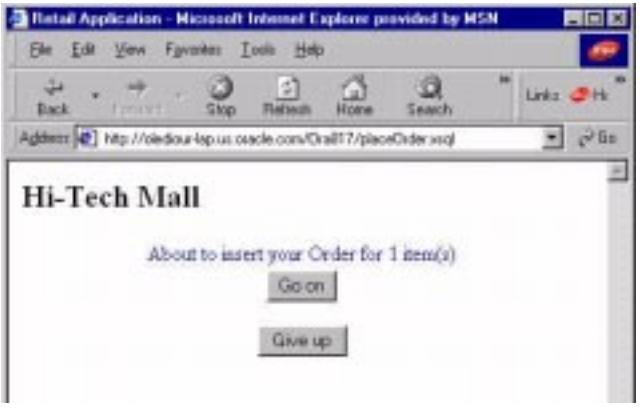
10. [Retailer] Click "Go On", or "Give Up". See Figure 13–14.

XSQL Script Example 4: Enter a Quantity — order.xsql

```
<?xml version="1.0"?>
<!--
 | This is the fourth script called.
 | It prompts you to enter a quantity.
 |
 | $Author: olediour@us $
 | $Revision: 1.1 $
 +-->
```

```
<?xml-stylesheet type="text/xsl" media="HandHTTP"     href="PP.xsl"?>
<?xml-stylesheet type="text/xsl" media="Mozilla"      href="HTML.xsl"?>

<order xmlns:xsql="urn:oracle-xsql"
       connection="retail"
       custId="000"
       prodId="000">
  <pageTitle>Hi-Tech Mall</pageTitle>
  <xsql:query tag-case      = "upper"
              rowset-element= ""
              row-element   = "cust">
    <![CDATA[
    select C.ID,
           C.NAME
    from CUSTOMER C
    where C.ID = '{@custId}'
    ]]>
    <xsql:no-rows-query>
      Select '{@custId}' as "unknown" from dual
    </xsql:no-rows-query>
  </xsql:query>

  <xsql:query tag-case="upper"
       rowset-element=""
       row-element="prod">
    <![CDATA[
        select I.ID,
               I.DESCRIPTION,
               I.PRICE,
               S.NAME
        from INVENTORY_ITEM I,
             SUPPLIER S
        where I.SUPPLIER_ID = S.ID and
              I.ID = '{@prodId}'
    ]]>
    <xsql:no-rows-query>
      Select '{@prodId}' as "unknown" from dual
    </xsql:no-rows-query>
  </xsql:query>

  <returnHome>index.xsql</returnHome>
</order>
```

Figure 13–14 [Retailer}: Click "Go On"

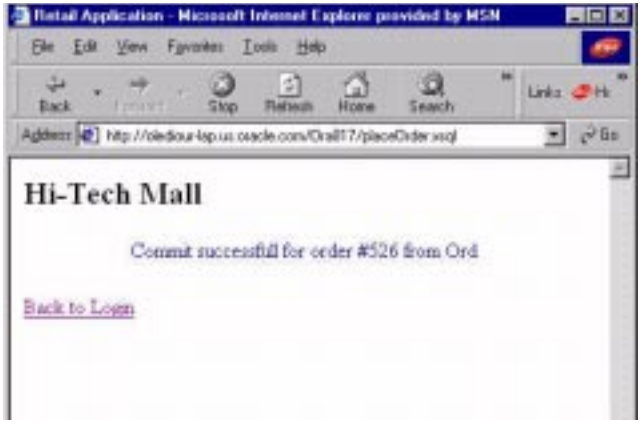


2 Retailer Places Order

The Retailer selects "Go On", then validates the order by selecting "Validate". See Figure 13–15 and Figure 13–16.

Figure 13–15 [Retailer]: Click "Validate"

Figure 13–16 [Retailer]: Commit Successful. Table Ord has Been Updated



3 "Validate" Commits the Transaction. Retailer Application Produces the XML Order

1. Once "Validate" is clicked, this triggers the main B2B process by means of the XSQL Servlet Action Handler. This is the end of client's interaction.

   The following scripts are executed by the B2B application (demo):

   - XSQL Script Example 5: Starts B2B Process — placeorder.xsql
   - Java Example 1: Action Handler Called by placeOrder.xsql — RetailActionHandler.java
   - Java Example 2: Maintains Session Context for RetailActionHandler.java — SessionHolder.java

XSQL Script Example 5: Starts B2B Process — placeorder.xsql

```
<?xml version="1.0"?>
<!--
 | This is the fifth and last, but not least, script called.
 | This script actually fires the whole B2B process.
 | It uses the Action Handler facility of XSQL Servlet.
 |
 | $Author: olediour@us $
 | $Revision: 1.1 $
 +-->
<?xml-stylesheet type="text/xsl" media="HandHTTP"     href="PP.xsl"?>
<?xml-stylesheet type="text/xsl" media="Mozilla"      href="HTML.xsl"?>

<placeOrder xmlns:xsql="urn:oracle-xsql"
            connection="retail"
            dbUrl     ="jdbc:oracle:thin:@atp-1.us.oracle.com:1521:ORCL"
            username  ="retailer"
            password  ="retailer"
            entity    ="Ord"
            operation ="insert"
            custId    =""
            ordId     =""
            prodId    =""
            qty       =""><
  <xsql:include-request-params/>
  <pageTitle>Hi-Tech Mall</pageTitle>
  <pageSeparator/>
```

```
    <xsql:action handler    ="B2BDemo.XSQLActionHandler.RetailActionHandler"
                 dbUrl      ="{@dbUrl}"
                 username   ="{@username}"
                 password   ="{@password}"
                 entity     ="{@entity}"
                 operation  ="{@operation}"
                 custId     ="{@custId}"
                 ordId      ="{@ordId}"
                 prodId     ="{@prodId}"
                 qty        ="{@qty}"/>
    <pageSeparator/>
    <bottomLinks>
      <aLink href="placeOrder.xsql?operation=rollback">Rollback</aLink>
    </bottomLinks>
    <returnHome>index.xsql</returnHome>
  </placeOrder>
```

Java Example 1: Action Handler Called by placeOrder.xsql — RetailActionHandler.java

Note: This example traverses almost 20 pages.

```
package B2BDemo.XSQLActionHandler;
/**
 * Action Handler called by the placeOrder.xsql script.
 * Actually fires the B2B process itself.
 * Uses SessionHolder to maintain transaction state.
 *
 * @see SessionHolder
 * @see placeOrder.xsql
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Corp.
 */
import oracle.xml.xsql.*;
import oracle.xml.xsql.actions.XSQLIncludeXSQLHandler;
import javax.servlet.http.*;
import javax.servlet.*;
import org.w3c.dom.*;

import java.sql.*;
import java.io.*;
```

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

import B2BDemo.AQUtil.*;
import B2BDemo.*;
import B2BDemo.XMLUtil.*;

public class RetailActionHandler extends XSQLActionHandlerImpl
{
  private static final boolean verbose   = false;
  private static final boolean debugFile = false;

  private Connection actionConnection = null;

  private String appUrl      = "";
  private String appUser     = "";
  private String appPassword = "";

  public static final String DBURL       = "dbUrl";
  public static final String USERNAME    = "username";
  public static final String PASSWORD    = "password";

  public static final String OPERATION   = "operation";

  public static final String ENTITY      = "entity";

  public static final String ORDID       = "ordId";
  public static final String ORDERDATE   = "orderDate";
  public static final String CONTACTNAME = "contactName";
  public static final String TRACKINGNO  = "trackingNo";
  public static final String STATUS      = "status";
  public static final String CUSTID      = "custId";

  public static final String QTY         = "qty";
  public static final String PRODID      = "prodId";

  public static final String SELECT      = "select";
  public static final String INSERT      = "insert";
  public static final String BEGIN       = "begin";
  public static final String COMMIT      = "commit";
  public static final String ROLLBACK    = "rollback";

  XSQLActionHandler nestedHandler = null;
  String operation = null;

  String entity       = null;
```

```
String ordId        = null;
String orderDate    = null;
String contactName  = null;
String trackingNo   = null;
String status       = null;
String custId       = null;
String qty          = null;
String prodId       = null;

HttpServletRequest     request  = null;
HttpServletResponse    response = null;
HttpSession            session  = null;

public void init(XSQLPageRequest xspRequest, Element action)
{
  super.init(xspRequest, action);
  // Retrieve the parameters

  if (verbose)
    System.out.println("init Action Handler.................");

  appUrl      = getAttributeAllowingParam(DBURL,    action);
  appUser     = getAttributeAllowingParam(USERNAME, action);
  appPassword = getAttributeAllowingParam(PASSWORD, action);

  operation = getAttributeAllowingParam(OPERATION, action);
  entity    = getAttributeAllowingParam(ENTITY, action);

  ordId       = getAttributeAllowingParam(ORDID,       action);
  orderDate   = getAttributeAllowingParam(ORDERDATE,   action);
  contactName = getAttributeAllowingParam(CONTACTNAME, action);
  trackingNo  = getAttributeAllowingParam(TRACKINGNO,  action);
  status      = getAttributeAllowingParam(STATUS,      action);
  custId      = getAttributeAllowingParam(CUSTID,      action);
  prodId      = getAttributeAllowingParam(PRODID,      action);
  qty         = getAttributeAllowingParam(QTY,         action);
  //
  if (verbose)
  {
    System.out.println("OrdID  > " + ordId);
    System.out.println("CustID > " + custId);
    System.out.println("ProdID > " + prodId);
  }

  final String HOLDER_NAME = "XSQLActionHandler.connection";
```

```
  try
  {
    if (xspRequest.getRequestType().equals("Servlet"))
    {
      XSQLServletPageRequest xspr = (XSQLServletPageRequest)xspRequest;
      HttpServletRequest req      = xspr.getHttpServletRequest();
      session = req.getSession(true); // true : Create if missing !!!
      if (verbose)
        System.out.println("Session Id = " + session.getId() + " - new : " +
                                                          session.isNew());
      SessionHolder sh = (SessionHolder) session.getValue(HOLDER_NAME);
      if (sh == null)
      {
        if (verbose)
          System.out.println("New SessionHandler > Getting connected at " +
                                                    (new java.util.Date()));
        actionConnection = getConnected(appUrl, appUser, appPassword);
        sh = new SessionHolder(actionConnection);
        session.putValue(HOLDER_NAME, sh);
      }
      actionConnection = sh.getConnection();
      if (verbose)
      {
        System.out.println("Reusing Connection at " + (new java.util.Date()) +
                           " - Opened at " + sh.getOpenDate().toString());
        System.out.println("Driver     : " +
                          actionConnection.getMetaData().getDriverName());
        System.out.println("SessionId  : " + session.getId());
        System.out.println("AutoCommit : " +
                                      actionConnection.getAutoCommit());
      }
    }
  }
  catch (Exception e)
  {
    System.err.println("Error in retrieving session context \n" + e);
    e.printStackTrace();
  }
}

// The result is the out parameter
public void handleAction(Node result) throws SQLException
{
  XSQLPageRequest xpr = getPageRequest();
  if (xpr.getRequestType().equals("Servlet"))
```

```
{
  // Get the servlet context and components
  XSQLServletPageRequest xspr = (XSQLServletPageRequest)xpr;
  request  = xspr.getHttpServletRequest();
  response = xspr.getHttpServletResponse();

  Document doc = null;

  // Display CLASSPATH
  XMLDocument myDoc = new XMLDocument();
  try
  {
    Element root = myDoc.createElement("root");
    myDoc.appendChild(root);

    Element cp = myDoc.createElement("ClassPath");
    root.appendChild(cp);

    // The text is a descendant of its node
    Node cpTxt = myDoc.createTextNode("text#");
    cpTxt.setNodeValue(System.getProperty("java.class.path"));
    cp.appendChild(cpTxt);

    Element e = myDoc.getDocumentElement();
    e.getParentNode().removeChild(e);
    result.appendChild(e);  // Append child to result before returning it.
  }
  catch (Exception e)
  {
    System.err.println("Building XMLDoc");
    e.printStackTrace();
  }
  try
  {
    // Add a node to hold operation value
    XMLDocument xmlDoc = new XMLDocument();
    Element elmt = xmlDoc.createElement("requiredOperation");
    xmlDoc.appendChild(elmt);
    Node theText = xmlDoc.createTextNode("text#");
    theText.setNodeValue(operation);
    elmt.appendChild(theText);
    // Append to result
    Element e = xmlDoc.getDocumentElement();
    e.getParentNode().removeChild(e);
    result.appendChild(e);  // Append child to result before returning it.
```

```
}
catch (Exception e)
{
  System.err.println("Building XMLDoc (2)");
  e.printStackTrace();
}

try
{
  // Dispatch
  if (operation.equals(SELECT))
 /* doc = manageSelect() */;
  else if (operation.equals(INSERT))
    doc = manageInsert();
  else if (operation.equals(BEGIN))
    doc = doBegin();
  else if (operation.equals(COMMIT))
    doc = doCommit();
  else if (operation.equals(ROLLBACK))
    doc = doRollback();
  else // Wrong operation
  {
    XMLDocument xmlDoc = new XMLDocument();
    Element elmt = xmlDoc.createElement("unknownOperation");
    xmlDoc.appendChild(elmt);
    Node theText = xmlDoc.createTextNode("text#");
    theText.setNodeValue(operation);
    elmt.appendChild(theText);
    // Append to result
    Element e = xmlDoc.getDocumentElement();
    e.getParentNode().removeChild(e);
    result.appendChild(e);  // Append child to result before returning it.
  }
}
catch (Exception ex)
{
// file://this.reportError(e);
  XMLDocument xmlDoc = new XMLDocument();
  Element elmt = xmlDoc.createElement("operationProblem");
  xmlDoc.appendChild(elmt);
  Node theText = xmlDoc.createTextNode("text#");
  theText.setNodeValue(ex.toString());
  elmt.appendChild(theText);
  // Append to result
  Element e = xmlDoc.getDocumentElement();
```

```
        e.getParentNode().removeChild(e);
        result.appendChild(e);  // Append child to result before returning it.
      }

      try
      {
        if (doc != null)
        {
          Element e = doc.getDocumentElement();
          e.getParentNode().removeChild(e);
          result.appendChild(e);  // Append child to result before returning it.
        }
      }
      catch (Exception e)
      {
        try
        {
          ServletOutputStream out = response.getOutputStream();
          out.println(e.toString());
        }
        catch (Exception ex) {}
      }
    }
    else  // Command line ?
    {
      System.out.println("Request type is [" + xpr.getRequestType() + "]");
    }
  }

/**
 * Removed because uselezss in this demo.
 *
  private Document manageSelect() throws Exception
  {
    Document doc = null;
    String cStmt = "";

    if (custId != null && custId.length() > 0)
      vo.setWhereClause("Customer_Id = '" + custId + "'");
    else
      vo.setWhereClause(null);
    vo.executeQuery();
    doc = data.getXMLDocument(); // Query implicitly executed !
    return doc;
  }
```

```
*/
  private Document manageInsert() throws Exception
  {
    Document doc = null;

    if (entity.equals("Ord"))
      doc = insertInOrd();
    else if (entity.equals("LineItem"))
      doc = insertInLine();
    else
    {
      doc = new XMLDocument();
      Element elmt = doc.createElement("operationQuestion");
      Attr attr = doc.createAttribute("opType");
      attr.setValue("insert");
      elmt.setAttributeNode(attr);
      doc.appendChild(elmt);
      Node txt = doc.createTextNode("text#");
      elmt.appendChild(txt);
      txt.setNodeValue("Don't know what to do with " + entity);
    }
    return doc;
  }

  private Document insertInOrd()
  {
    Document doc = null;
    if (custId == null || custId.length() == 0)
    {
      doc = new XMLDocument();
      Element elmt = doc.createElement("operationProblem");
      Attr attr = doc.createAttribute("opType");
      attr.setValue("OrdInsert");
      elmt.setAttributeNode(attr);
      doc.appendChild(elmt);
      Node txt = doc.createTextNode("text#");
      elmt.appendChild(txt);
      txt.setNodeValue("Some element(s) missing for ord insert (custId)");
    }
    else
    {
      String seqStmt = "select Ord_Seq.nextVal from dual";
      String seqVal = "";
      try
      {
```

```
  Statement stmt = actionConnection.createStatement();
  ResultSet rSet = stmt.executeQuery(seqStmt);
  while (rSet.next())
    seqVal = rSet.getString(1);
  rSet.close();
  stmt.close();
}
catch (SQLException e)
{
  System.err.println("Error reading ORD_SEQ Sequence : " + e.toString());
}
//                                    1          2         3            4
String cStmt = "insert into ORD values (?, sysdate, ?, 'AX' || ?,
                                                          'Pending', ?)";
try
{
  if (verbose)
    System.out.println("Inserting Order # " + seqVal);
  PreparedStatement pStmt = actionConnection.prepareStatement(cStmt);
  pStmt.setString(1, seqVal);
  pStmt.setString(2, "Ora817");  // Default value !
  pStmt.setString(3, seqVal);
  pStmt.setString(4, custId);
  pStmt.execute();
  pStmt.close();
/**
  try
  {
    Statement stmt = actionConnection.createStatement();
    ResultSet rSet = stmt.executeQuery("SELECT * FROM ORD WHERE ID = " +
                                                              seqVal);
    int i = 0;
    while (rSet.next())
      i++;
    if (verbose)
      System.out.println(i + " record found for " + seqVal);
    rSet.close();
    stmt.close();
  }
  catch (SQLException e)
  {
    System.err.println("Error : " + e.toString());
  }
 */
  doc = new XMLDocument();
```

```
Element elmt = doc.createElement("operationResult");
Attr attr = doc.createAttribute("opType");
attr.setValue("insert");
elmt.setAttributeNode(attr);

attr = doc.createAttribute("Step");
attr.setValue(entity);
elmt.setAttributeNode(attr);

doc.appendChild(elmt);
Node txt = doc.createTextNode("text#");
elmt.appendChild(txt);
txt.setNodeValue("About to insert your Order for " + qty + " item(s)");

Element nextElmt = doc.createElement("nextStep");
elmt.appendChild(nextElmt);
attr = doc.createAttribute("Label");
attr.setValue("Go on");
nextElmt.setAttributeNode(attr);

attr = doc.createAttribute("Action");
nextElmt.setAttributeNode(attr);
attr.setValue("placeOrder.xsql");
Element pList = doc.createElement("prmList");
nextElmt.appendChild(pList);
// viewobject
Element prm = doc.createElement("prm");
pList.appendChild(prm);
attr = doc.createAttribute("name");
attr.setValue("entity");
prm.setAttributeNode(attr);
attr = doc.createAttribute("value");
attr.setValue("LineItem");
prm.setAttributeNode(attr);
// custId
prm = doc.createElement("prm");
pList.appendChild(prm);
attr = doc.createAttribute("name");
attr.setValue("custId");
prm.setAttributeNode(attr);
attr = doc.createAttribute("value");
attr.setValue(custId);
prm.setAttributeNode(attr);
// prodId
prm = doc.createElement("prm");
```

```
pList.appendChild(prm);
attr = doc.createAttribute("name");
attr.setValue("prodId");
prm.setAttributeNode(attr);
attr = doc.createAttribute("value");
attr.setValue(prodId);
prm.setAttributeNode(attr);
// qty
prm = doc.createElement("prm");
pList.appendChild(prm);
attr = doc.createAttribute("name");
attr.setValue("qty");
prm.setAttributeNode(attr);
attr = doc.createAttribute("value");
attr.setValue(qty);
prm.setAttributeNode(attr);
// ordId
prm = doc.createElement("prm");
pList.appendChild(prm);
attr = doc.createAttribute("name");
attr.setValue("ordId");
prm.setAttributeNode(attr);
attr = doc.createAttribute("value");
attr.setValue(seqVal);
prm.setAttributeNode(attr);

nextElmt = doc.createElement("nextStep");
elmt.appendChild(nextElmt);
attr = doc.createAttribute("Label");
attr.setValue("Give up");
nextElmt.setAttributeNode(attr);

attr = doc.createAttribute("Action");
nextElmt.setAttributeNode(attr);
attr.setValue("placeOrder.xsql");
pList = doc.createElement("prmList");
nextElmt.appendChild(pList);
// viewobject
prm = doc.createElement("prm");
pList.appendChild(prm);
attr = doc.createAttribute("name");
attr.setValue("operation");
prm.setAttributeNode(attr);
attr = doc.createAttribute("value");
attr.setValue("rollback");
```

```
      prm.setAttributeNode(attr);
    }
    catch (Exception e)
    {
      doc = new XMLDocument();
      Element elmt = doc.createElement("operationProblem");
      Attr attr = doc.createAttribute("opType");
      attr.setValue("insert");
      elmt.setAttributeNode(attr);

      attr = doc.createAttribute("Step");
      attr.setValue(entity);
      elmt.setAttributeNode(attr);

      doc.appendChild(elmt);
      Node txt = doc.createTextNode("text#");
      elmt.appendChild(txt);
      txt.setNodeValue(e.toString());
      if (verbose)
        System.out.println("Error : " + e.toString());
      Element prm = doc.createElement("parameters");
      elmt.appendChild(prm);
      // ID
      Element prmVal = doc.createElement("ID");
      prm.appendChild(prmVal);
      txt = doc.createTextNode("text#");
      prmVal.appendChild(txt);
      txt.setNodeValue(ordId);
      // CUSTOMER_ID
      prmVal = doc.createElement("CUSTOMER_ID");
      prm.appendChild(prmVal);
      txt = doc.createTextNode("text#");
      prmVal.appendChild(txt);
      txt.setNodeValue(custId);
    }
  }
  return doc;
}

private Document insertInLine()
{
  Document doc = null;
  if (custId == null || custId.length() == 0 ||
      qty == null || qty.length() == 0 ||
      prodId == null || prodId.length() == 0 ||
```

```
    ordId == null || ordId.length() == 0)
{
  doc = new XMLDocument();
  Element elmt = doc.createElement("operationProblem");
  Attr attr = doc.createAttribute("opType");
  attr.setValue("lineInsert");
  elmt.setAttributeNode(attr);
  doc.appendChild(elmt);
  Node txt = doc.createTextNode("text#");
  elmt.appendChild(txt);
  txt.setNodeValue("Some element(s) missing for line insert (" +
                   ((custId == null || custId.length() == 0)?"custId ":"") +
                   ((qty == null || qty.length() == 0)?"qty ":"") +
                   ((prodId == null || prodId.length() == 0)?"prodId ":"") +
                   ((ordId == null || ordId.length() == 0)?"ordId ":"") +")"
                   );

  Element subElmt = doc.createElement("custId");
  elmt.appendChild(subElmt);
  txt = doc.createTextNode("text#");
  subElmt.appendChild(txt);
  txt.setNodeValue(custId);

  subElmt = doc.createElement("qty");
  elmt.appendChild(subElmt);
  txt = doc.createTextNode("text#");
  subElmt.appendChild(txt);
  txt.setNodeValue(qty);

  subElmt = doc.createElement("prodId");
  elmt.appendChild(subElmt);
  txt = doc.createTextNode("text#");
  subElmt.appendChild(txt);
  txt.setNodeValue(prodId);

  subElmt = doc.createElement("ordId");
  elmt.appendChild(subElmt);
  txt = doc.createTextNode("text#");
  subElmt.appendChild(txt);
  txt.setNodeValue(ordId);
}
else
{
  if (verbose)
    System.out.println("Inserting line : Ord>" + ordId + ", Prod>" + prodId
```

```
                                                          + ", Qty>" + qty);
/**
  try
  {
    Statement stmt = actionConnection.createStatement();
    ResultSet rSet = stmt.executeQuery("SELECT * FROM ORD WHERE ID = " +
                                                              ordId);
    int i = 0;
    while (rSet.next())
      i++;
    System.out.println(i + " record found for " + ordId);
    rSet.close();
    stmt.close();
  }
  catch (SQLException e)
  {
    System.err.println("Error : " + e.toString());
  }
 */
  String cStmt = "insert into line_item values (Line_item_seq.nextVal, ?, ?,
                                                                ?, 0)";
  try
  {
    PreparedStatement pStmt = actionConnection.prepareStatement(cStmt);
    pStmt.setString(1, qty);
    pStmt.setString(2, prodId);
    pStmt.setString(3, ordId);
    pStmt.execute();
    pStmt.close();

    doc = new XMLDocument();
    Element elmt = doc.createElement("operationResult");
    Attr attr = doc.createAttribute("opType");
    attr.setValue("insert");
    elmt.setAttributeNode(attr);

    attr = doc.createAttribute("Step");
    attr.setValue(entity);
    elmt.setAttributeNode(attr);

    doc.appendChild(elmt);
    Node txt = doc.createTextNode("text#");
    elmt.appendChild(txt);
    txt.setNodeValue("Insert Successful");
```

```
Element nextElmt = doc.createElement("nextStep");
elmt.appendChild(nextElmt);
attr = doc.createAttribute("Label");
attr.setValue("Validate");
nextElmt.setAttributeNode(attr);

attr = doc.createAttribute("Action");
nextElmt.setAttributeNode(attr);
attr.setValue("placeOrder.xsql");
Element pList = doc.createElement("prmList");
nextElmt.appendChild(pList);
// operation
Element prm = doc.createElement("prm");
pList.appendChild(prm);
attr = doc.createAttribute("name");
attr.setValue("operation");
prm.setAttributeNode(attr);
attr = doc.createAttribute("value");
attr.setValue("commit");
prm.setAttributeNode(attr);
// ordId
prm = doc.createElement("prm");
pList.appendChild(prm);
attr = doc.createAttribute("name");
attr.setValue("ordId");
prm.setAttributeNode(attr);
attr = doc.createAttribute("value");
attr.setValue(ordId);
prm.setAttributeNode(attr);

nextElmt = doc.createElement("nextStep");
elmt.appendChild(nextElmt);
attr = doc.createAttribute("Label");
attr.setValue("Cancel");
nextElmt.setAttributeNode(attr);

attr = doc.createAttribute("Action");
nextElmt.setAttributeNode(attr);
attr.setValue("placeOrder.xsql");
pList = doc.createElement("prmList");
nextElmt.appendChild(pList);
// operation
prm = doc.createElement("prm");
pList.appendChild(prm);
attr = doc.createAttribute("name");
```

```
    attr.setValue("operation");
    prm.setAttributeNode(attr);
    attr = doc.createAttribute("value");
    attr.setValue("rollback");
    prm.setAttributeNode(attr);
  }
  catch (Exception e)
  {
    if (verbose)
      System.out.println("Error when inserting " + e.toString());

    doc = new XMLDocument();
    Element elmt = doc.createElement("operationProblem");
    Attr attr = doc.createAttribute("opType");
    attr.setValue("insert");
    elmt.setAttributeNode(attr);

    attr = doc.createAttribute("Step");
    attr.setValue(entity);
    elmt.setAttributeNode(attr);
    doc.appendChild(elmt);

    Node txt = doc.createTextNode("text#");
    elmt.appendChild(txt);
    txt.setNodeValue(e.toString());

    Element prm = doc.createElement("parameters");
    elmt.appendChild(prm);
    // ID
    Element prmVal = doc.createElement("ORD_ID");
    prm.appendChild(prmVal);
    txt = doc.createTextNode("text#");
    prmVal.appendChild(txt);
    txt.setNodeValue(ordId);
    // QTY
    prmVal = doc.createElement("QTY");
    prm.appendChild(prmVal);
    txt = doc.createTextNode("text#");
    prmVal.appendChild(txt);
    txt.setNodeValue(qty);
    // ITEM_ID
    prmVal = doc.createElement("ITEM_ID");
    prm.appendChild(prmVal);
    txt = doc.createTextNode("text#");
    prmVal.appendChild(txt);
```

```
        txt.setNodeValue(prodId);
      }
    }
    return doc;
  }

  private Document doCommit() throws Exception
  {
    Document doc = null;
    actionConnection.commit();

    doc = new XMLDocument();
    Element elmt = doc.createElement("operationResult");
    Attr attr = doc.createAttribute("opType");
    attr.setValue("commit");
    elmt.setAttributeNode(attr);
    doc.appendChild(elmt);
    Node txt = doc.createTextNode("dummy");
    elmt.appendChild(txt);
    txt.setNodeValue("Commit successfull for order #" + ordId + " from " +
entity);

    if (ordId != null && ordId.length() > 0)
    {
      // Generate XML Document to send to AQ
      // Start from Ord with OrdId value -

      AQWriter aqw = null;

      aqw = new AQWriter(AppCste.AQuser,
                         AppCste.AQpswd,
                         AppCste.AQDBUrl,
                         "AppOne_QTab",
                         "AppOneMsgQueue");

      String doc2send = XMLGen.returnDocument(actionConnection, ordId);
      // sending XMLDoc in the Queue
      try
      {
        if (verbose)
          System.out.println("Doc : " + doc2send);
        if (debugFile)
        {
          BufferedWriter bw = new BufferedWriter(new FileWriter("debug.txt"));
          bw.write("Rows in " + entity);
```

```
        bw.write(doc2send);
        bw.flush();
        bw.close();
      }
    }
    catch (Exception ex) {}

    aqw.writeQ(new B2BMessage(MessageHeaders.APP_A,
                              MessageHeaders.APP_B,
                              MessageHeaders.NEW_ORDER,
                              doc2send));
    aqw.flushQ();  // Commit !
  }

  return doc;
}

private Document doRollback() throws Exception
{
  Document doc = null;
  actionConnection.rollback();

  doc = new XMLDocument();
  Element elmt = doc.createElement("operationResult");
  Attr attr = doc.createAttribute("opType");
  attr.setValue("rollback");
  elmt.setAttributeNode(attr);
  doc.appendChild(elmt);
  Node txt = doc.createTextNode("dummy");
  elmt.appendChild(txt);
  txt.setNodeValue("Rollback successfull");

  return doc;
}

private Document doBegin() throws Exception
{
  Document doc = null;
  actionConnection.setAutoCommit(false);

  doc = new XMLDocument();
  Element elmt = doc.createElement("operationResult");
  Attr attr = doc.createAttribute("opType");
  attr.setValue("begin");
  elmt.setAttributeNode(attr);
```

```
    doc.appendChild(elmt);
    Node txt = doc.createTextNode("dummy");
    elmt.appendChild(txt);
    txt.setNodeValue("Begin successfull");

    return doc;
  }

  private static Connection getConnected(String connURL,
                                         String userName,
                                         String password)
  {
    Connection conn = null;
    try
    {
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
      conn = DriverManager.getConnection(connURL, userName, password);
      conn.setAutoCommit(false);
    }
    catch (Exception e)
    {
      System.err.println(e);
      System.exit(1);
    }
    return conn;
  }
}
```

Java Example 2: Maintains Session Context for RetailActionHandler.java — SessionHolder.java

```
// Copyright (c) 2000 Oracle Corporation
package B2BDemo.XSQLActionHandler;
/**
 * Used to maintain the connection context from the XSQL Action Handler.
 * Also closes the connection when servlet expires.
 *
 * @see RetailActionHandler
 */
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;

public class SessionHolder implements HttpSessionBindingListener
```

```
{
  private Connection c;
  private java.util.Date d = null;

  public SessionHolder(Connection conn)
  {
    System.out.println("New SessionHandler");
    this.c = conn;
    this.d = new java.util.Date();
  }

  public Connection getConnection()
  {
    return this.c;
  }

  public java.util.Date getOpenDate()
  {
    return this.d;
  }

  public void valueBound(HttpSessionBindingEvent event)
  {
    System.out.println("\nvalueBound ! " + event.getName() + "\nat " + (new
          java.util.Date()) + "\nfor " + event.getSession().getId());
  }

  public void valueUnbound(HttpSessionBindingEvent event)
  {
    System.out.println("\nvalueUnbound ! " + event.getName() + "\nat " + (new
                  java.util.Date()) + "\nfor " + event.getSession().getId());
    event.getSession().removeValue("XSQLActionHandler.connection");
    if (this.c != null)
    {
      try { this.c.close(); }
      catch (Exception e)
      {
        System.out.println("Problem when closing the connection from " +
                           event.getName() +
                           " for " +
                           event.getSession().getId() +
                           " :\n" +
                           e);
      }
    }
```

```
  }
}
```

4 AQ Broker-Transformer Transforms XML Document According to Supplier's Format

1. AQ Broker-Transformer application is alerted that an XML order is pending.
2. An XML document containing the details of your order has been produced using the XML-SQL Utility. This document has been sent to the AQ Broker-Transformer for propagation, using Advanced Queuing.

   The AQ Broker application knows the following, based on its Stylesheet table:

   - Who it comes from: Retailer
   - Who it goes to: Supplier
   - What its for: NEW ORDER

   These elements are used to select the correct stylesheet from Stylesheet table. XSL-T Processor processes the transformation. See Figure 13–17.

Scripts:

- `MessageBroker. java` calls `BrokerThread.java` which calls
  - `BrokerThread.java` calls `AQReader.java` and `AQWriter.java`

    `AQReader.java` and `AQWriter.java` both use `B2BMessages.java` for their message structure.

Figure 13–17 [AQ Broker]: Viewing the retailer.bat, broker.bat, and supplier.bat Consoles (1 of 3)

3. Hit [Return] in the AQ Broker Console.
4. The correct stylesheet is found inside the Stylesheets table according to contents in the AppFrom, AppTo, and Op columns. The XSL Transformation proceeds using the selected stylesheet. We now have a reformatted XML document ready for the Supplier.

Note: Here XML + XSL = XML

---

5. Again hit [Return] in the AQ Broker Console. See Figure 13–18. The broker.bat screen changes as it has been transforming. The result is obtained after the XSL-T transformation.

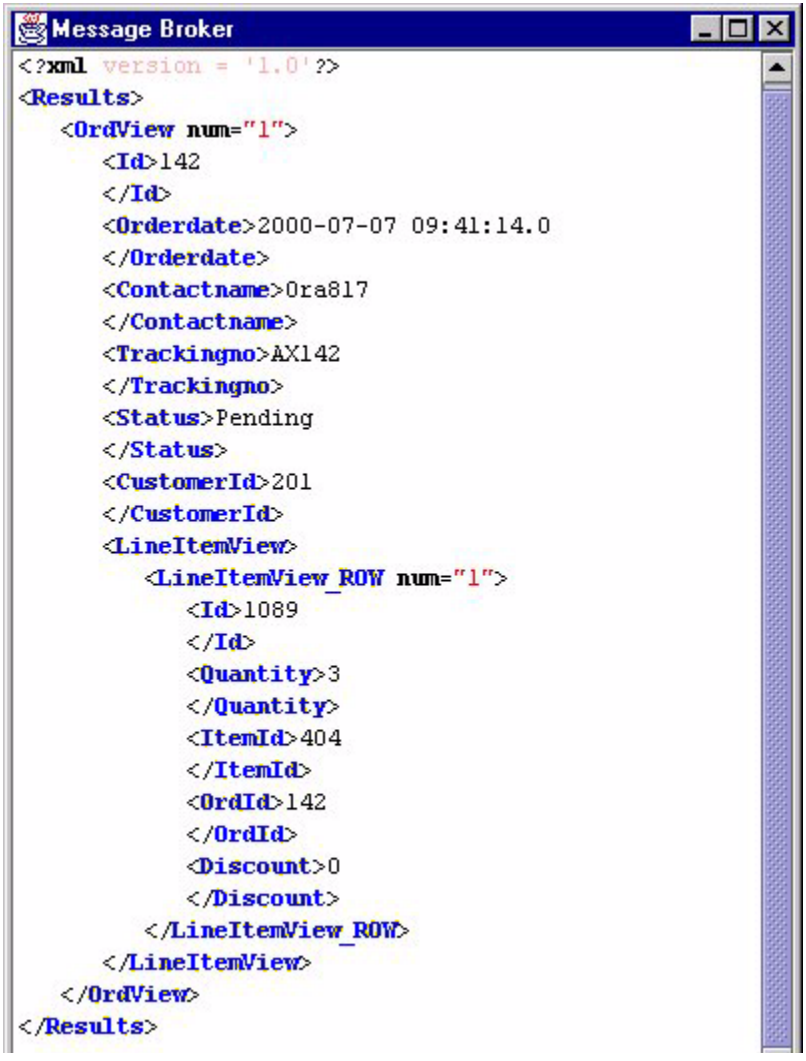
See the "AQ Broker-Transformer and Advanced Queuing Scripts" section for code listings that run the AQ Broker-Transformer and the Advanced Queuing process.

Figure 13–18 [AQ Broker]: Viewing the retailer.bat, broker.bat, and supplier.bat Consoles (2 of 3)

The newly reformatted XML document is sent to the Supplier by means of Advanced Queuing [WRITE].

Note: The AQ Broker and Supplier .bat screens should look the same as both applications are processing the same XML document at this moment.

Figure 13–19 Sample XML Document Output From AQ Broker-Transformer



5 Supplier Application Parses the XML Document and Inserts the Order into the Supplier Database

1. The XML document is received by the Supplier application. It now needs to be parsed for the data it contains, and this data is then inserted into the database.
2. Hit [Return] in the Supplier's Console. See Figure 13–20.

Figure 13–20 [AQ Broker]: Viewing the retailer.bat, broker.bat, and supplier.bat Consoles (3 of 3)

6a Supplier Application Alerts Supplier of Pending Order

1. The document is processed and the data is inserted. The Supplier application Watcher program sends a wake up message that an order is pending! See Figure 13–21.
2. Click OK in the Supplier's Watcher dialog box. See Figure 13–22.

Scripts:

- `SupplierWatcher.java` calls `SupplierFramer.java`

Figure 13–21 The Supplier Application Alerts Supplier of Pending Order: "Wake Up!"

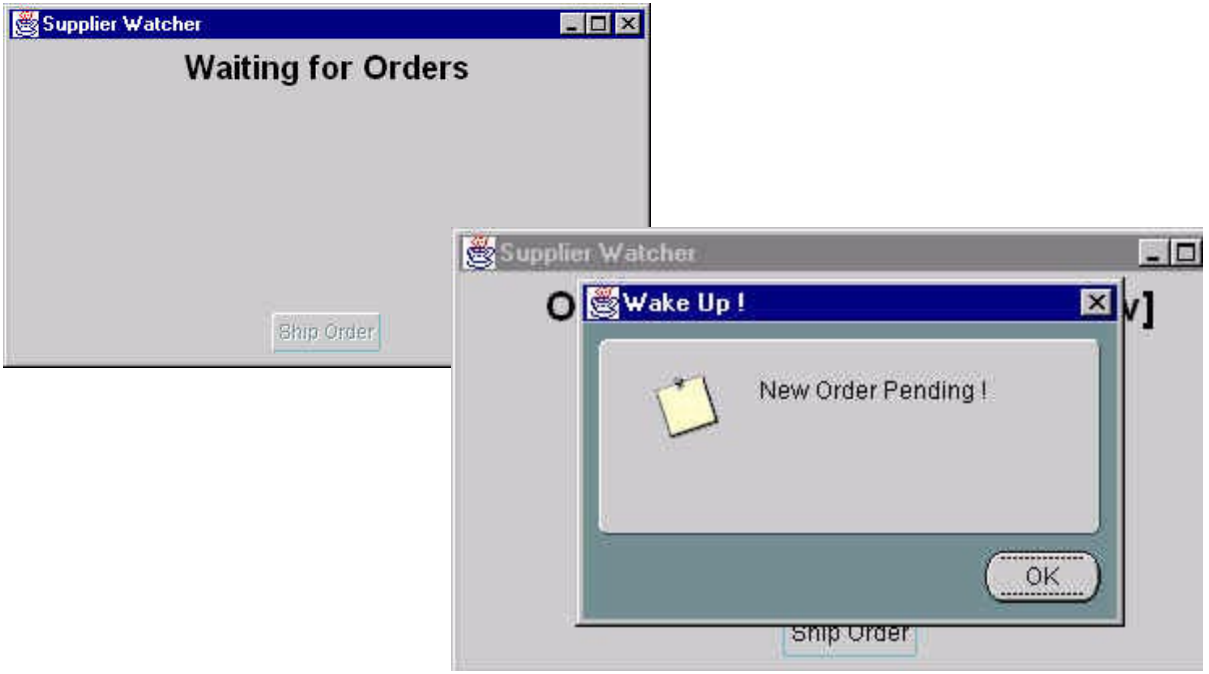


Figure 13–22 [Supplier]: retail.bat, broker.bat, and supplier.bat Consoles: After Clicking OK to Wake Up

6b Supplier Decides to Ship the Product(s) to the Retailer

1. You, the supplier, decide to ship this order. Click "Ship Order" in the dialog box. See Figure 13–23 and Figure 13–24.

Scripts: Still using `SupplierWatcher.java`.

Figure 13–23 [Supplier]: Decides to Ship the Order

Figure 13–24 [Supplier]: Viewing the retailer.bat, broker.bat, and supplier.bat Consoles on "Ship Order"

6c Supplier Application Generates a New XML Message to Send to AQ Broker

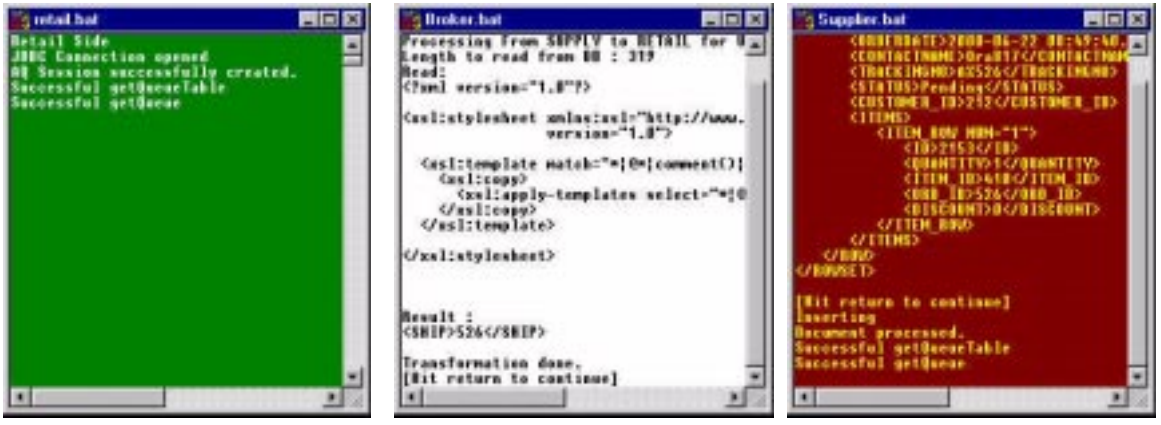
1. The Supplier application shipping order generates a new message, sent to the broker.
2. Hit [Return] in the Broker's Console. See Figure 13–25.

Figure 13–25 [Supplier]: retailer.bat, broker.bat, and supplier.bat Consoles - Form New XML Document

7 AQ Broker-Transformer Transforms XML Order into Retailer's Format

1. As in Step 4, a stylesheet is chosen from AQ Broker-Transformer database and applied to the XML order document to produce a reformatted XML document.
2. Hit [Return] in the Broker's Console. See Figure 13–26.

Figure 13–26 [AQ Broker]: retailer.bat, broker.bat, and supplier.bat Consoles - Reformat XML Document



3. The document is sent to the Retailer application.
4. Hit [Return] in Retailer's Console. See Figure 13–27. This parses the XML order.

Figure 13–27 [AQ Broker]: retailer.bat, broker.bat, and supplier.bat Consoles - Sending XMLMessage

8 Retailer Application Updates the Ord Table and Displays the New Order Status to Retailer

1. Retailer application updates the Retailer database "Pending " status with the new "shipped" order status information. The Ord table is updated.
2. This information is viewed by the Retailer from any device. The status is seen as "Shipped". See Figure 13–28.

Scripts:

`UpdateMaster.java`. This receives the message and parses it.

Figure 13–28 [Retailer]: retailer.bat, broker.bat, and supplier.bat Consoles - Updates Status to Shipped

That's it!

To Stop the B2B XML Application

To stop the B2B XML application (demo), run Java Example 3: stopQ.bat.

Java Example 3: stopQ.bat

```
@echo off
@echo stopping all Qs
D:\jdev31\java\bin\java -mx50m -classpath
"D:\xml817\references\Ora817DevGuide;
```

```
D:\jdev31\lib\jdev-rt.zip;
D:\jdev31\jdbc\lib\oracle8.1.6\classes111.zip;
D:\jdev31\lib\connectionmanager.zip;
D:\jdev31\lib;D:\jdev31\lib\oraclexsql.jar;
D:\jdev31\lib\oraclexmlsql.jar;
D:\jdev31\lib\xmlparserv2_2027.jar;
D:\jdev31\jfc\lib\swingall.jar;
D:\jdev31\jswdk-1.0.1\lib\servlet.jar;
D:\Ora8i\rdbms\jlib\aqapi11.jar;
D:\Ora8i\rdbms\jlib\aqapi.jar;
D:\XMLWorkshop\xmlcomp.jar;
D:\jdev31\java\lib\classes.zip"  B2BDemo.AQUtil.StopAllQueues
```

Check Your Order Status Directly Using vieworder.sql

To view your order status directly from the database run this SQL script.

```
set ver off
select O.ID as "Order#",
   O.OrderDate as "Order Date",
   O.Status as "Status"
   From ORD O,
     CUSTOMER C
   Where O.CUSTOMER_ID = C.ID and
     Upper(C.NAME) = Upper('&CustName');
```

Java Examples - Calling Sequence

The following list provides the Java examples' calling sequence. The .java extension for each file has been omitted. The notation "<---" implies "calls", for example, AQReader <----- B2BMessage implies that AQReader calls B2BMessage.

- AQReader <---- B2BMessage
- AQWriter <---- B2BMessage
- UpdateMaster
  - <---- AQReader <----B2BMessage
  - <---- B2BMessage
  - <----- MessageHeaders
  - XMLFrame
- SupplierWatcher
  - <---- SupplierFrame
    - <---- AQReader <---- B2BMessage
    - <----XML2DMLv2 <---- TableInDocument
    - <---- TableInDocument
    - <---- AQWriter <---- B2BMessage
    - <---- B2BMessage
    - <---- MessageHeaders
  - <---- XMLFrame
- MessageBroker
  - <---- AppCste
  - <---- BrokerThread
    - <---- XSLTWrapper
    - <---- AQWriter <---- B2BMessage
    - <---- AQReader <---- B2BMessage
  - <---- AQReader <---- B2BMessage
  - <---- AQWriter <---- B2BMessage

- <---- XMLFrame (called by MessageBroker)
- RetailActionHandler <---- SessionHolder

XSL and XSL Management Scripts

To prevent over complicating the listing of examples in the section, "Running the B2B XML Application: Detailed Procedure", the XSL examples are listed separately.

- XSL Stylesheet Example 1: Converts Results to HTML — html.xsl
- XSL Stylesheet Example 2: Converts Results for Palm Pilot Browser — pp.xsl
- Java Example 3: Stylesheet Management— GUIInterface.java
- Java Example 4: GUIInterface_AboutBoxPanel.java
- Java Example 5: GUIStylesheet.java

XSL Stylesheet Example 1: Converts Results to HTML — html.xsl

```
<?xml version="1.0"?>
<!--
| $Author: olediour@us $
| $Date: 04-May-2000
| xsl for html
| $Revision: 1.1 $
+-->

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">
  <xsl:output media-type="text/html" method="html" encoding="ISO-8859-1"/>

  <xsl:template match="/">
   <html>
    <head>
      <title>Retail Application</title>
    </head>
    <body>
      <xsl:if test="//pageTitle">
        <h2><xsl:value-of select="//pageTitle"/></h2>
      </xsl:if>
      <xsl:choose>
        <xsl:when test="loginResult">
          <xsl:apply-templates select="loginResult"/>
        </xsl:when>
        <xsl:when test="index">
          <xsl:apply-templates select="index"/>
        </xsl:when>
        <xsl:when test="inventory">
```

```
          <xsl:apply-templates select="inventory"/>
        </xsl:when>
        <xsl:when test="order">
          <xsl:apply-templates select="order"/>
        </xsl:when>
        <xsl:when test="placeOrder">
          <xsl:apply-templates select="placeOrder"/>
        </xsl:when>
        <xsl:otherwise>
           <p align="center">
             <h3>This kind of XML Document cannot be processed...</h3>
           </p>
        </xsl:otherwise>
      </xsl:choose>
    </body>
  </html>
</xsl:template>

<xsl:template match="loginResult">
 <xsl:if test="ROWSET/ROW/unknown">
  <table width="98%">
  <tr>
  <td bgcolor="yellow" align="center">
  <xsl:value-of select="ROWSET/ROW/unknown"/> is not allowed to log in !</td>
  </tr>
  </table>
  </xsl:if>
  <xsl:if test="ROWSET/ROW/NAME">
   <p align="center">
    <h2>Welcome <xsl:value-of select="ROWSET/ROW/NAME"/> !</h2>
   </p>
   <p align="center">
    <a>
    <xsl:attribute name="href">
<xsl:value-of select="nextStep"/>?custId=<xsl:value-of select="ROWSET/ROW/ID"/>
</xsl:attribute>
        Please enter the Mall !
      </a>
    </p>
  </xsl:if>
  <p>
    <a><xsl:attribute name="href"><xsl:value-of
select="returnHome"/></xsl:attribute>Back to Login</a>
  </p>
</xsl:template>
```

```
<xsl:template match="index">
  <xsl:for-each select="form">
    <center>
      <form>
 <xsl:attribute name="action"><xsl:value-of select="./@action"/></xsl:attribute>
 <xsl:attribute name="method"><xsl:value-of select="./@method"/></xsl:attribute>
     <xsl:if test="./field">
       <table width="98%" border="1">
         <xsl:for-each select="./field">
           <tr>
             <td align="right"><xsl:value-of select="./@prompt"/></td>
             <td>
              <input>
                <xsl:choose>
                <xsl:when test="./@type = 'text'">
                <xsl:attribute name="type">text</xsl:attribute>
                </xsl:when>
                </xsl:choose>
                <xsl:attribute name="name">
                <xsl:value-of select="./@name"/></xsl:attribute>
                </input>
                </td>
                </tr>
                </xsl:for-each>
       </table>
     </xsl:if>
    <xsl:if test="./button">
    <p>
    <xsl:for-each select="./button">
      <input>
        <xsl:choose>
         <xsl:when test="./@type = 'submit'">
          <xsl:attribute name="type">submit</xsl:attribute>
         </xsl:when>
        </xsl:choose>
       <xsl:attribute name="value">
       <xsl:value-of select="./@label"/>
      </xsl:attribute>
     </input>
    </xsl:for-each>
    </p>
    </xsl:if>
 </form>
     </center>
```

```
    </xsl:for-each>
  </xsl:template>

  <xsl:template match="inventory">
    <h2>This is the Mart content</h2>
    <table>
      <tr>
        <th>Prod #</th>
        <th>Product</th>
        <th>Price</th>
        <th>Supplied by</th>
      </tr>
      <xsl:for-each select="form/theMart/ROWSET/ROW">
        <tr>
          <td><xsl:value-of select="ID"/></td>
          <td>
            <a>
              <xsl:attribute name="href">
                <xsl:value-of
select="../../../../form/@action"/>?custId=<xsl:value-of
select="../../../../form/hiddenFields/custId"/>&amp;prodId=<xsl:value-of
select="ID"/>
              </xsl:attribute>
              <xsl:value-of select="DESCRIPTION"/>
            </a>
          </td>
          <td><xsl:value-of select="PRICE"/></td>
          <td><xsl:value-of select="NAME"/></td>
        </tr>
      </xsl:for-each>
    </table>
    <p>
      <a><xsl:attribute name="href"><xsl:value-of
select="returnHome"/></xsl:attribute>Back to Login</a>
    </p>
  </xsl:template>

  <xsl:template match="order">
    <center>
      <h2>Thank you <xsl:value-of select="CUST/NAME"/> for shopping with us
!</h2>
      <hr/>
      <h2>Please enter the quantity</h2>
      <form action="placeOrder.xsql" method="post">
        <input type="hidden" name="prodId">
```

```
          <xsl:attribute name="value">
          <xsl:value-of select="PROD/ID"/>
        </xsl:attribute>
        </input>
        <input type="hidden" name="custId">
          <xsl:attribute name="value">
          <xsl:value-of select="CUST/ID"/></xsl:attribute>
        </input>
        <table border="1">
          <tr>
            <td colspan="2"><xsl:value-of select="PROD/DESCRIPTION"/>
                at $<xsl:value-of select="PROD/PRICE"/> each
                supplied by <xsl:value-of select="PROD/NAME"/></td>
          </tr>
          <tr>
            <td align="right">Quantity</td>
            <td><input type="text" name="qty"/></td>
          </tr>
        </table>
        <p><input type="submit" value="Place Order"/></p>
      </form>
    </center>
    <p>
      <a><xsl:attribute name="href">
      <xsl:value-of select="returnHome"/>
     </xsl:attribute>Back to Login</a>
    </p>
  </xsl:template>

  <xsl:template match="placeOrder">
    <xsl:if test="operationResult">
      <table width="98%">
        <tr><td align="center">
        <font color="navy">
        <xsl:value-of select="operationResult/text()"/>
        </font></td></tr>
        <tr>
          <td align="center">
            <xsl:for-each select="operationResult/nextStep">
              <form method="post">
                <xsl:attribute name="action"><xsl:value-of
select="./@Action"/></xsl:attribute>
                <xsl:if test="prmList">
                  <xsl:for-each select="prmList/prm">
                    <input type="hidden">
```

```
                    <xsl:attribute name="name"><xsl:value-of
select="./@name"/></xsl:attribute>
                    <xsl:attribute name="value"><xsl:value-of
select="./@value"/></xsl:attribute>
                  </input>
                </xsl:for-each>
              </xsl:if>
              <input type="submit">
                <xsl:attribute name="value"><xsl:value-of
select="./@Label"/></xsl:attribute>
              </input>
            </form>
          </xsl:for-each>
        </td>
      </tr>
    </table>
  </xsl:if>
  <xsl:if test="xsql-error">
    <table width="98%">
      <tr><td><xsl:value-of select="xsql-error/@action"/></td></tr>
      <tr><td><xsl:value-of select="xsql-error/statement"/></td></tr>
      <tr><td><xsl:value-of select="xsql-error/message"/></td></tr>
    </table>
  </xsl:if>
  <xsl:if test="operationProblem">
    <table width="98%">
      <tr>
        <td colspan="2" align="center">
          <font color="red"><b><xsl:value-of
select="operationProblem/text()"/></b></font>
        </td>
      </tr>
      <xsl:for-each select="operationProblem/parameters/*">
        <tr>
          <td align="right"><xsl:value-of select="name()"/></td>
          <td align="left"><xsl:value-of select="."/></td>
        </tr>
      </xsl:for-each>
    </table>
  </xsl:if>
  <xsl:if test="bottomLinks">
    <xsl:choose>
      <xsl:when test="operationResult">
      </xsl:when>
      <xsl:otherwise>
```

```
          <p align="center">
            <xsl:for-each select="bottomLinks/aLink">
              [<a><xsl:attribute name="href"><xsl:value-of
select="./@href"/></xsl:attribute><xsl:value-of select="."/></a>]
            </xsl:for-each>
          </p>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:if>
    <xsl:choose>
      <xsl:when test="operationResult/nextStep">
      </xsl:when>
      <xsl:otherwise>
        <xsl:if test="returnHome">
          <p>
            <a><xsl:attribute name="href"><xsl:value-of
select="returnHome"/></xsl:attribute>Back to Login</a>
          </p>
        </xsl:if>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

</xsl:stylesheet>
```

XSL Stylesheet Example 2: Converts Results for Palm Pilot Browser — pp.xsl

```
<?xml version="1.0"?>
<!--
| $Author: olediour@us $
| $Date: 04-May-2000
| xsl for html (Palm Pilot, HandWeb browser)
| $Revision: 1.1 $
+-->

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">
  <xsl:output media-type="text/html" method="html" encoding="ISO-8859-1"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>Retail Application</title>
      </head>
```

```
      <body>
        <xsl:if test="//pageTitle">
          <h2><xsl:value-of select="//pageTitle"/></h2>
        </xsl:if>
        <xsl:choose>
          <xsl:when test="loginResult">
            <xsl:apply-templates select="loginResult"/>
          </xsl:when>
          <xsl:when test="index">
            <xsl:apply-templates select="index"/>
          </xsl:when>
          <xsl:when test="inventory">
            <xsl:apply-templates select="inventory"/>
          </xsl:when>
          <xsl:when test="order">
            <xsl:apply-templates select="order"/>
          </xsl:when>
          <xsl:when test="placeOrder">
            <xsl:apply-templates select="placeOrder"/>
          </xsl:when>
          <xsl:otherwise>
             <p align="center">
               <h3>This kind of XML Document cannot be processed...</h3>
             </p>
          </xsl:otherwise>
        </xsl:choose>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="loginResult">
    <xsl:if test="ROWSET/ROW/unknown">
      <table width="98%">
        <tr><td bgcolor="yellow" align="center"><xsl:value-of
select="ROWSET/ROW/unknown"/> is not allowed to log in !</td></tr>
      </table>
    </xsl:if>
    <xsl:if test="ROWSET/ROW/NAME">
      <p align="center">
        <h2>Welcome <xsl:value-of select="ROWSET/ROW/NAME"/> !</h2>
      </p>
      <p align="center">
        <a>
          <xsl:attribute name="href"><xsl:value-of
select="nextStep"/>?custId=<xsl:value-of
```

```
select="ROWSET/ROW/ID"/></xsl:attribute>
          Please enter the Mall !
        </a>
      </p>
    </xsl:if>
    <p>
      <a><xsl:attribute name="href"><xsl:value-of
select="returnHome"/></xsl:attribute>Back to Login</a>
    </p>
  </xsl:template>

  <xsl:template match="index">
    <xsl:for-each select="form">
      <center>
        <form>
          <xsl:attribute name="action"><xsl:value-of
select="./@action"/></xsl:attribute>
          <xsl:attribute name="method"><xsl:value-of
select="./@method"/></xsl:attribute>
          <xsl:if test="./field">
            <table width="98%" border="1">
              <xsl:for-each select="./field">
                <tr>
                  <td align="right"><xsl:value-of select="./@prompt"/></td>
                  <td>
                    <input>
                      <xsl:choose>
                        <xsl:when test="./@type = 'text'">
                          <xsl:attribute name="type">text</xsl:attribute>
                        </xsl:when>
                        </xsl:choose>
                       <xsl:attribute name="name"><xsl:value-of
select="./@name"/></xsl:attribute>
                    </input>
                  </td>
                </tr>
              </xsl:for-each>
            </table>
          </xsl:if>
          <xsl:if test="./button">
            <p>
              <xsl:for-each select="./button">
                <input>
                  <xsl:choose>
                    <xsl:when test="./@type = 'submit'">
```

```
                    <xsl:attribute name="type">submit</xsl:attribute>
                  </xsl:when>
                </xsl:choose>
                <xsl:attribute name="value"><xsl:value-of
select="./@label"/></xsl:attribute>
              </input>
            </xsl:for-each>
          </p>
        </xsl:if>
      </form>
    </center>
  </xsl:for-each>
 </xsl:template>

 <xsl:template match="inventory">
   <h2>This is the Mart content</h2>
   <xsl:for-each select="form/theMart/ROWSET/ROW">
     <xsl:value-of select="ID"/>
     <xsl:text> </xsl:text>
     <form method="post">
       <xsl:attribute name="action">
         <xsl:value-of select="../../../../form/@action"/>
       </xsl:attribute>
       <input type="hidden" name="custId">
         <xsl:attribute name="value"><xsl:value-of
select="../../../../form/hiddenFields/custId"/></xsl:attribute>
       </input>
       <input type="hidden" name="prodId">
         <xsl:attribute name="value"><xsl:value-of
select="ID"/></xsl:attribute>
       </input>
       <input type="submit">
         <xsl:attribute name="value"><xsl:value-of
select="DESCRIPTION"/></xsl:attribute>
       </input>
     </form>
     <xsl:text> @ $</xsl:text><xsl:value-of select="PRICE"/><xsl:text>
each</xsl:text>
     <xsl:text> Supplied by </xsl:text><xsl:value-of select="NAME"/>
     <br/>
   </xsl:for-each>
   <p>
     <a><xsl:attribute name="href"><xsl:value-of
select="returnHome"/></xsl:attribute>Back to Login</a>
   </p>
```

```
  </xsl:template>

  <xsl:template match="order">
    <center>
      <h2>Thank you <xsl:value-of select="CUST/NAME"/> for shopping with us
!</h2>
      <hr/>
      <h2>Please enter the quantity</h2>
      <form action="placeOrder.xsql" method="post">
        <input type="hidden" name="prodId">
          <xsl:attribute name="value"><xsl:value-of
select="PROD/ID"/></xsl:attribute>
        </input>
        <input type="hidden" name="custId">
          <xsl:attribute name="value"><xsl:value-of
select="CUST/ID"/></xsl:attribute>
        </input>
        <p>
          <xsl:value-of select="PROD/DESCRIPTION"/>
                at $<xsl:value-of select="PROD/PRICE"/> each
                supplied by <xsl:value-of select="PROD/NAME"/>
          <br/>
            Quantity :
          <br/>
          <input type="text" name="qty"/>
        </p>
        <p><input type="submit" value="Place Order"/></p>
      </form>
    </center>
    <p>
      <a><xsl:attribute name="href"><xsl:value-of
select="returnHome"/></xsl:attribute>Back to Login</a>
    </p>
  </xsl:template>

  <xsl:template match="placeOrder">
    <xsl:if test="operationResult">
      <center>
        <xsl:value-of select="operationResult/text()"/>
        <br/>
        <xsl:for-each select="operationResult/nextStep">
          <form method="post">
            <xsl:attribute name="action"><xsl:value-of
select="./@Action"/></xsl:attribute>
            <xsl:if test="prmList">
```

```
              <xsl:for-each select="prmList/prm">
                <input type="hidden">
                  <xsl:attribute name="name"><xsl:value-of
select="./@name"/></xsl:attribute>
                  <xsl:attribute name="value"><xsl:value-of
select="./@value"/></xsl:attribute>
                </input>
              </xsl:for-each>
            </xsl:if>
            <input type="submit">
              <xsl:attribute name="value"><xsl:value-of
select="./@Label"/></xsl:attribute>
            </input>
          </form>
        </xsl:for-each>
      </center>
    </xsl:if>
    <xsl:if test="operationProblem">
      <table width="98%">
        <tr><td align="center"><font color="red"><xsl:value-of
select="operationProblem"/></font></td></tr>
      </table>
    </xsl:if>
    <xsl:if test="bottomLinks">
      <xsl:choose>
        <xsl:when test="operationResult">
        </xsl:when>
        <xsl:otherwise>
          <p align="center">
            <xsl:for-each select="bottomLinks/aLink">
              [<a><xsl:attribute name="href"><xsl:value-of
select="./@href"/></xsl:attribute><xsl:value-of select="."/></a>]
            </xsl:for-each>
          </p>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:if>
    <xsl:choose>
      <xsl:when test="operationResult/nextStep">
      </xsl:when>
      <xsl:otherwise>
        <xsl:if test="returnHome">
          <p>
            <a><xsl:attribute name="href"><xsl:value-of
select="returnHome"/></xsl:attribute>Back to Login</a>
```

```
          </p>
        </xsl:if>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

</xsl:stylesheet>
```

Java Example 3: Stylesheet Management— GUIInterface.java

This script creates and manages the GUI and stylesheets used in the B2B XML application.

```
package B2BDemo.StyleSheetUtil;
/**
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

import java.sql.*;
import java.util.*;
// needed for new CLOB and BLOB classes
import oracle.sql.*;
import oracle.jdbc.driver.*;
import java.beans.*;
import javax.swing.event.*;

import B2BDemo.*;
import B2BDemo.XMLUtil.*;

public class GUInterface extends JFrame
{
  private boolean lite = false;  // Use O8iLite
  private boolean inserting = false;

  private final static int UPDATE = 1;
  private final static int INSERT = 2;

  private final static int ENTER_QUERY = 1;
  private final static int EXEC_QUERY  = 2;

  int queryState = ENTER_QUERY;
```

```
String sqlStmt  = "Select APPFROM, " +
                  "       APPTO, " +
                  "       OP, " +
                  "       XSL " +
                  "From styleSheets";

private static String connURL  = AppCste.AQDBUrl;
private static String userName = AppCste.AQuser;
private static String password = AppCste.AQpswd;
private Connection conn = null;

private Vector recVect = null;
int currRec = 0;
XslRecord thisRecord = null;

BorderLayout borderLayout1 = new BorderLayout();
JPanel jPanel1 = new JPanel();
JMenuBar menuBar1 = new JMenuBar();
JMenu menuFile = new JMenu();
JMenuItem menuFileExit = new JMenuItem();
JMenu menuHelp = new JMenu();
JMenuItem menuHelpAbout = new JMenuItem();
JLabel statusBar = new JLabel();
JToolBar toolBar = new JToolBar();
JButton buttonOpen = new JButton();
JButton buttonClose = new JButton();
JButton buttonHelp = new JButton();
ImageIcon imageOpen;
ImageIcon imageClose;
ImageIcon imageHelp;
JPanel jPanel2 = new JPanel();
BorderLayout borderLayout2 = new BorderLayout();
JButton firstButton = new JButton();
JPanel jPanel3 = new JPanel();
JPanel jPanel4 = new JPanel();
BorderLayout borderLayout3 = new BorderLayout();
BorderLayout borderLayout4 = new BorderLayout();
JPanel jPanel5 = new JPanel();
JTextField fromAppValue = new JTextField();
JLabel fromApp = new JLabel();
JPanel jPanel6 = new JPanel();
BorderLayout borderLayout5 = new BorderLayout();
JLabel jLabel2 = new JLabel();
JScrollPane jScrollPane1 = new JScrollPane();
```

```
  JTextArea XSLStyleSheet = new JTextArea();
  JButton previousButton = new JButton();
  JButton nextButton = new JButton();
  JButton lastButton = new JButton();
  JButton validateButton = new JButton();
  GridLayout gridLayout1 = new GridLayout();
  JLabel toApp = new JLabel();
  JTextField toAppValue = new JTextField();
  JLabel operationLabel = new JLabel();
  JTextField opValue = new JTextField();
  JButton newButton = new JButton();
  JButton deleteButton = new JButton();
  JButton queryButton = new JButton();

  public GUInterface()
  {
    super();
    try
    {
      jbInit();
      buttonOpen.setIcon(imageOpen);
      buttonClose.setIcon(imageClose);
      buttonHelp.setIcon(imageHelp);
    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
  }

  private void getConnected() throws Exception
  {
    try
    {
      if (lite)
      {
        Class.forName("oracle.lite.poljdbc.POLJDBCDriver");
        conn = DriverManager.getConnection("jdbc:Polite:POLite", "system",
"manager");
      }
      else
      {
        Class.forName ("oracle.jdbc.driver.OracleDriver");
        conn = DriverManager.getConnection (connURL, userName, password);
      }
```

```
  }
  catch (Exception e)
  {
    System.err.println("Get connected failed : " + e);
    throw e;
  }
}

private void jbInit() throws Exception
{
  if (conn == null)
  {
    try { getConnected(); }
    catch (Exception e)
    {
      JOptionPane.showMessageDialog(null, e.toString(),
                                    "Connection",
                                    JOptionPane.ERROR_MESSAGE);
      System.exit(1);
    }
  }
  imageOpen = new ImageIcon(GUInterface.class.getResource("openfile.gif"));
  imageClose = new ImageIcon(GUInterface.class.getResource("closefile.gif"));
  imageHelp = new ImageIcon(GUInterface.class.getResource("help.gif"));
  this.setTitle("Style Sheets Management");
  this.getContentPane().setLayout(borderLayout1);
  this.setSize(new Dimension(511, 526));
  jPanel1.setLayout(borderLayout2);
  menuFile.setText("File");
  menuFileExit.setText("Exit");
  menuFileExit.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      fileExit_ActionPerformed(e);
    }
  });
  menuHelp.setText("Help");
  menuHelpAbout.setText("About");
  menuHelpAbout.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      helpAbout_ActionPerformed(e);
    }
  });
  statusBar.setText("Initializing...");
  buttonOpen.setToolTipText("Open File");
  buttonClose.setToolTipText("Validate modifications");
```

```
buttonHelp.setToolTipText("About Style Sheet Manager");
firstButton.setText("<<");
jPanel5.setLayout(gridLayout1);
fromApp.setText("From Application :");
fromApp.setHorizontalAlignment(SwingConstants.RIGHT);
jLabel2.setText("XSL Style Sheet");
previousButton.setText("<");
nextButton.setText(">");
lastButton.setText(">>");
validateButton.setText("Validate");
gridLayout1.setRows(4);
toApp.setText("To Application : ");
toApp.setHorizontalAlignment(SwingConstants.RIGHT);
operationLabel.setText("Operation : ");
operationLabel.setHorizontalAlignment(SwingConstants.RIGHT);
jPanel6.setLayout(borderLayout5);
jPanel4.setLayout(borderLayout4);
jPanel3.setLayout(borderLayout3);
menuFile.add(menuFileExit);
menuBar1.add(menuFile);
menuHelp.add(menuHelpAbout);
menuBar1.add(menuHelp);
this.setJMenuBar(menuBar1);
this.getContentPane().add(statusBar, BorderLayout.SOUTH);
toolBar.add(buttonOpen);
toolBar.add(buttonClose);
toolBar.add(buttonHelp);
this.getContentPane().add(toolBar, BorderLayout.NORTH);
this.getContentPane().add(jPanel1, BorderLayout.CENTER);
jPanel1.add(jPanel2, BorderLayout.SOUTH);
jPanel2.add(queryButton, null);
jPanel2.add(newButton, null);
jPanel2.add(firstButton, null);
jPanel2.add(previousButton, null);
jPanel2.add(nextButton, null);
jPanel2.add(lastButton, null);
jPanel2.add(validateButton, null);
jPanel2.add(deleteButton, null);
jPanel1.add(jPanel3, BorderLayout.CENTER);
jPanel3.add(jPanel4, BorderLayout.NORTH);
jPanel3.add(jPanel5, BorderLayout.SOUTH);
jPanel5.add(fromApp, null);
jPanel5.add(fromAppValue, null);
jPanel5.add(toApp, null);
jPanel5.add(toAppValue, null);
```

```
jPanel5.add(operationLabel, null);
jPanel5.add(opValue, null);
jPanel3.add(jPanel6, BorderLayout.CENTER);
jPanel6.add(jLabel2, BorderLayout.NORTH);
jPanel6.add(jScrollPane1, BorderLayout.CENTER);
jScrollPane1.getViewport().add(XSLStyleSheet, null);

//
statusBar.setText("Connected...");
// Building Vector of record.
queryButton.setText("Enter Query");
queryButton.setActionCommand("query");
queryButton.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    queryButton_actionPerformed(e);
  }
});
buttonClose.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    buttonClose_actionPerformed(e);
  }
});
deleteButton.setText("Delete");
deleteButton.setToolTipText("Delete the current record");
deleteButton.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    deleteButton_actionPerformed(e);
  }
});
newButton.setText("New");
newButton.setToolTipText("Create a new record");
newButton.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    newButton_actionPerformed(e);
  }
});
validateButton.setToolTipText("Validate your modifications");
```

```
opValue.setEditable(false);
toAppValue.setEditable(false);
fromAppValue.setEditable(false);
validateButton.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    validateButton_actionPerformed(e);
  }
});
lastButton.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    lastButton_actionPerformed(e);
  }
});
firstButton.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    firstButton_actionPerformed(e);
  }
});
previousButton.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    previousButton_actionPerformed(e);
  }
});
nextButton.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    nextButton_actionPerformed(e);
  }
});
lastButton.setActionCommand("last");
lastButton.setToolTipText("Last record");
nextButton.setActionCommand("next");
nextButton.setToolTipText("Next record");
previousButton.setActionCommand("previous");
previousButton.setToolTipText("Previous record");
firstButton.setActionCommand("first");
```

```
  firstButton.setToolTipText("First record");

  // Execute query and build vector
  executeQuery(sqlStmt);

  updateStatusBar();
}

void executeQuery(String theSqlStmt)
{
  recVect = new Vector();
  try
  {
    Statement stmt = conn.createStatement();
    ResultSet rSet = stmt.executeQuery(theSqlStmt);
    CLOB clob = null;
    while (rSet.next())
    {
      clob = ((OracleResultSet)rSet).getCLOB(4);
      String strLob = dumpClob(conn, clob);
      XslRecord xslRecord = new XslRecord(rSet.getString(1),
                                          rSet.getString(2),
                                          rSet.getString(3),
                                          strLob);
      recVect.addElement(xslRecord);
    }
    rSet.close();
    stmt.close();
    // Populate form with first record
    firstButton.setEnabled(false);
    previousButton.setEnabled(false);
    nextButton.setEnabled(false);
    lastButton.setEnabled(false);
    if (recVect.size() > 0)
    {
      currRec = 1;
      displayRecord(currRec);
    }
    if (recVect.size() > 1)
    {
      nextButton.setEnabled(true);
      lastButton.setEnabled(true);
    }
  }
  catch (Exception e)
```

```
  {
    JOptionPane.showMessageDialog(null, e.toString(),
                                  "Executing request",
                                  JOptionPane.ERROR_MESSAGE);
    System.exit(1);
  }
}

void displayRecord(int rnk)
{
  XslRecord xslRecord = (XslRecord)recVect.elementAt(rnk-1);
  thisRecord = new XslRecord(xslRecord.FROM,
                             xslRecord.TO,
                             xslRecord.TASK,
                             xslRecord.XSL);
  XSLStyleSheet.setText(xslRecord.XSL);
  fromAppValue.setText(xslRecord.FROM);
  toAppValue.setText(xslRecord.TO);
  opValue.setText(xslRecord.TASK);

  XSLStyleSheet.requestFocus();
  XSLStyleSheet.setCaretPosition(0);

  // Buttons
  firstButton.setEnabled(false);
  previousButton.setEnabled(false);
  nextButton.setEnabled(false);
  lastButton.setEnabled(false);
  if (rnk > 1)
  {
    firstButton.setEnabled(true);
    previousButton.setEnabled(true);
  }
  if (rnk < recVect.size())
  {
    nextButton.setEnabled(true);
    lastButton.setEnabled(true);
  }
}

void updateStatusBar()
{
  statusBar.setText("Ready for " + recVect.size() + " records");
}
```

```
  private static String dumpClob(Connection conn, CLOB clob) throws Exception
  {
    String returnString = "";

    OracleCallableStatement cStmt1 = (OracleCallableStatement) conn.prepareCall
("begin ? := dbms_lob.getLength (?); end;");
    OracleCallableStatement cStmt2 = (OracleCallableStatement) conn.prepareCall
("begin dbms_lob.read (?, ?, ?, ?); end;");

    cStmt1.registerOutParameter (1, Types.NUMERIC);
    cStmt1.setCLOB (2, clob);
    cStmt1.execute ();

    long length = cStmt1.getLong (1);
    long i = 0;
    int chunk = 80;

    while (i < length)
    {
      cStmt2.setCLOB (1, clob);
      cStmt2.setLong (2, chunk);
      cStmt2.registerOutParameter (2, Types.NUMERIC);
      cStmt2.setLong (3, i + 1);
      cStmt2.registerOutParameter (4, Types.VARCHAR);
      cStmt2.execute ();

      long read_this_time = cStmt2.getLong (2);
      returnString += cStmt2.getString (4);
   // System.out.print ("Read " + read_this_time + " chars: ");
   // System.out.println (string_this_time);
      i += read_this_time;
    }
    cStmt1.close ();
    cStmt2.close ();
    return returnString;
  }

  static void fillClob (Connection conn, CLOB clob, String str) throws
SQLException
  {
    OracleCallableStatement cStmt =
      (OracleCallableStatement) conn.prepareCall ("begin dbms_lob.write (?, ?,
?, ?); end;");

    int i = 0;
```

```
      int chunk = 80;
      int length = str.length();
      long c, ii;

      System.out.println("Length: " + length + "\n" + str);
      while (i < length)
      {
        cStmt.setClob (1, clob);
        c = chunk;
        cStmt.setLong (2, c);
        ii = i + 1;
        cStmt.setLong (3, ii);
        cStmt.setString (4, str.substring(i, i + chunk));
        cStmt.execute ();
        i += chunk;
        if (length - i < chunk)
          chunk = length - i;
      }
      cStmt.close ();
    }

    void fileExit_ActionPerformed(ActionEvent e)
    {
      if (conn != null)
      {
        try { conn.close(); } catch (Exception ex) {}
      }
      System.exit(0);
    }

    void helpAbout_ActionPerformed(ActionEvent e)
    {
      JOptionPane.showMessageDialog(this, new GUInterface_AboutBoxPanel1(),
"About", JOptionPane.PLAIN_MESSAGE);
    }

    void nextButton_actionPerformed(ActionEvent e)
    {
      checkRecordChange();
      currRec++;
      displayRecord(currRec);
    }

    void previousButton_actionPerformed(ActionEvent e)
    {
```

```
  checkRecordChange();
  currRec--;
  displayRecord(currRec);
}

void firstButton_actionPerformed(ActionEvent e)
{
  checkRecordChange();
  currRec = 1;
  displayRecord(currRec);
}

void lastButton_actionPerformed(ActionEvent e)
{
  checkRecordChange();
  currRec = recVect.size();
  displayRecord(currRec);
}

void validateButton_actionPerformed(ActionEvent e)
{
  validateRec();
}

void validateRec()
{
  thisRecord = new XslRecord(fromAppValue.getText(),
                             toAppValue.getText(),
                             opValue.getText(),
                             XSLStyleSheet.getText());
  if (saveChanges(thisRecord, (inserting?INSERT:UPDATE)))
    JOptionPane.showMessageDialog(null, "All right!");
}

void deleteRec()
{
  thisRecord = new XslRecord(fromAppValue.getText(),
                             toAppValue.getText(),
                             opValue.getText(),
                             XSLStyleSheet.getText());
  String sqlStmt  = "delete styleSheets where fromApp = ? and " +
                    "                         toApp   = ? and " +
                    "                         op      = ?";
  try
  {
```

```
      PreparedStatement pStmt = conn.prepareStatement(sqlStmt);
      pStmt.setString(1, thisRecord.FROM);
      pStmt.setString(2, thisRecord.TO);
      pStmt.setString(3, thisRecord.TASK);
      pStmt.execute();
      conn.commit();
      System.out.println("Deleted !");
      pStmt.close();
      // Delete from vector...
      recVect.removeElementAt(currRec - 1);
      updateStatusBar();
      if (currRec >= recVect.size())
        currRec--;
      displayRecord(currRec);
      JOptionPane.showMessageDialog(null, "All right!");
    }
    catch (SQLException sqlE)
    {
      JOptionPane.showMessageDialog(null, sqlE.toString(),
                                    "Deleting record",
                                    JOptionPane.ERROR_MESSAGE);
    }
    catch (Exception e)
    {
      JOptionPane.showMessageDialog(null, e.toString(),
                                    "Deleting record",
                                    JOptionPane.ERROR_MESSAGE);
    }
  }

  void checkRecordChange()
  {
    thisRecord = new XslRecord(fromAppValue.getText(),
                               toAppValue.getText(),
                               opValue.getText(),
                               XSLStyleSheet.getText());
    if (!thisRecord.equals((XslRecord)recVect.elementAt(currRec-1)))
    {
      int result = JOptionPane.showConfirmDialog(null, "Record has changed\nDo
you want to save the modifications ?");
      if (result == JOptionPane.YES_OPTION)
      {
        saveChanges(thisRecord, UPDATE);
        JOptionPane.showMessageDialog(null, "All right!");
      }
```

```
    }
  }

  boolean saveChanges(XslRecord rec,
                      int operation)
  {
    boolean ret = true;
    if (operation == this.UPDATE)
    {
      String theSqlStmt = "update styleSheets set xsl = ? where appFrom = ? and
appTo = ? and op = ?";
      try
      {
        PreparedStatement pStmt = conn.prepareStatement(theSqlStmt);
        pStmt.setString(1, rec.XSL);
        pStmt.setString(2, rec.FROM);
        pStmt.setString(3, rec.TO);
          pStmt.setString(4, rec.TASK);
        pStmt.execute();
        conn.commit();
        System.out.println("Updated !");
        pStmt.close();
        // Reinsert in vector...
        recVect.setElementAt(rec, currRec - 1);
      }
      catch (SQLException sqlE)
      {
        JOptionPane.showMessageDialog(null, sqlE.toString(),
                                      "Saving record",
                                      JOptionPane.ERROR_MESSAGE);
        ret = false;
      }
    }
    else
    {
      System.out.println("Inserting new record");
      String sqlStmt  = "insert into styleSheets " +
                        "           ( appFrom,   " +
                        "             appTo,     " +
                        "             op,        " +
                        "             xsl        " +
                        "           ) values     " +
                        "           (?, ?, ?, ?)";
      String sqlGetLob = "select xsl from styleSheets " +
                         "where appFrom = ? and " +
```

```
                    "      appTo   = ? and " +
                    "      op      = ?";
  try
  {
    PreparedStatement pStmt = conn.prepareStatement(sqlStmt);
    pStmt.setString(1, rec.FROM);
    pStmt.setString(2, rec.TO);
    pStmt.setString(3, rec.TASK);
    pStmt.setString(4, ""); // Null in the LOB, will be filled later
    pStmt.execute();
    System.out.println("Inserted !");
    pStmt.close();

    PreparedStatement fillLOBStmt = conn.prepareStatement(sqlGetLob);
    fillLOBStmt.setString(1, rec.FROM);
    fillLOBStmt.setString(2, rec.TO);
    fillLOBStmt.setString(3, rec.TASK);
    ResultSet lobRSet = fillLOBStmt.executeQuery();
    while (lobRSet.next())
    {
      CLOB clob = ((OracleResultSet)lobRSet).getCLOB(1);
      fillClob(conn, clob, rec.XSL);
    }
    conn.commit();

    // Add in vector...
    recVect.addElement(rec);
    currRec = recVect.size();
    displayRecord(currRec);
  }
  catch (SQLException sqlE)
  {
    JOptionPane.showMessageDialog(null, sqlE.toString(),
                                  "Inserting record",
                                  JOptionPane.ERROR_MESSAGE);
    ret = false;
  }

  inserting = false;

  fromAppValue.setEditable(false);
  toAppValue.setEditable(false);
  opValue.setEditable(false);
}
updateStatusBar();
```

```
  return ret;
}

void buttonClose_actionPerformed(ActionEvent e)
{
  validateRec();
}

void newButton_actionPerformed(ActionEvent e)
{
  fromAppValue.setEditable(true);
  toAppValue.setEditable(true);
  opValue.setEditable(true);
  inserting = true;
  XSLStyleSheet.setText("");
  fromAppValue.setText("");
  toAppValue.setText("");
  opValue.setText("");
}

void deleteButton_actionPerformed(ActionEvent e)
{
  deleteRec();
}

void queryButton_actionPerformed(ActionEvent e)
{
  if (queryState == ENTER_QUERY)
  {
    queryState = EXEC_QUERY;
    queryButton.setText("Execute Query");
    fromAppValue.setEditable(true);
    toAppValue.setEditable(true);
    opValue.setEditable(true);

    XSLStyleSheet.setEditable(false);
    statusBar.setText("Entering query");
    XSLStyleSheet.setText("");
    fromAppValue.setText("");
    toAppValue.setText("");
    opValue.setText("");

    newButton.setEnabled(false);
    firstButton.setEnabled(false);
    previousButton.setEnabled(false);
```

```
      nextButton.setEnabled(false);
      lastButton.setEnabled(false);
      validateButton.setEnabled(false);
      deleteButton.setEnabled(false);
    }
    else
    {
      queryState = ENTER_QUERY;
      queryButton.setText("Enter Query");
      statusBar.setText("Executing query");

      fromAppValue.setEditable(false);
      toAppValue.setEditable(false);
      opValue.setEditable(false);
      XSLStyleSheet.setEditable(true);

      newButton.setEnabled(true);
      firstButton.setEnabled(true);
      previousButton.setEnabled(true);
      nextButton.setEnabled(true);
      lastButton.setEnabled(true);
      validateButton.setEnabled(true);
      deleteButton.setEnabled(true);

      // Execute query
      String stmt = sqlStmt;
      boolean firstCondition = true;
      if (fromAppValue.getText().length() > 0)
      {
        stmt += ((firstCondition?" where ":" and ") + "fromApp like '" +
fromAppValue.getText() + "' ");
        firstCondition = false;
      }
      if (toAppValue.getText().length() > 0)
      {
        stmt += ((firstCondition?" where ":" and ") + "toApp like '" +
toAppValue.getText() + "' ");
        firstCondition = false;
      }
      if (opValue.getText().length() > 0)
      {
        stmt += ((firstCondition?" where ":" and ") + "op like '" +
opValue.getText() + "' ");
        firstCondition = false;
      }
```

```
      executeQuery(stmt);
      updateStatusBar();
      displayRecord(currRec);
    }
  }
}
```

Java Example 4: GUIInterface_AboutBoxPanel.java

```
package B2BDemo.StyleSheetUtil;
/**
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
import java.awt.*;
import javax.swing.*;
import javax.swing.border.*;
import oracle.jdeveloper.layout.*;

public class GUInterface_AboutBoxPanel1 extends JPanel
{
  JLabel jLabel1 = new JLabel();
  JLabel jLabel2 = new JLabel();
  JLabel jLabel3 = new JLabel();
  JLabel jLabel4 = new JLabel();
  GridBagLayout gridBagLayout1 = new GridBagLayout();
  Border border1 = new EtchedBorder();


  public GUInterface_AboutBoxPanel1()
  {
    try
    {
      jbInit();
    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
  }

  private void jbInit() throws Exception
  {
   jLabel1.setText("Stored Style Sheets management.");
   jLabel2.setText("Olivier LE DIOURIS");
```

```
   jLabel3.setText("Copyright (c) 1999");
   jLabel4.setText("Oracle Corp.");
   this.setLayout(gridBagLayout1);
   this.setBorder(border1);
   this.add(jLabel1, new GridBagConstraints2(0, 0, 1, 1, 0.0, 0.0,
GridBagConstraints.WEST, GridBagConstraints.NONE, new Insets(5,5,0,5),0,0));
    this.add(jLabel2, new GridBagConstraints2(0, 1, 1, 1, 0.0, 0.0,
    GridBagConstraints.WEST, GridBagConstraints.NONE, new Insets(0,5,0,5),0,0));
    this.add(jLabel3, new GridBagConstraints2(0, 2, 1, 1, 0.0, 0.0,
    GridBagConstraints.WEST, GridBagConstraints.NONE, new Insets(0,5,0,5),0,0));
    this.add(jLabel4, new GridBagConstraints2(0, 3, 1, 1, 0.0, 0.0,
    GridBagConstraints.WEST, GridBagConstraints.NONE, new Insets(0,5,5,5),0,0));
  }
}
```

Java Example 5: GUIStylesheet.java

```
package B2BDemo.StyleSheetUtil;
/**
 * A grapical utility to manipulate the stylesheets stored in the database,
 * in the AQ Schema. The stylsheets will be used to transform the incoming
 * document into the outgoing one.
 *
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
import java.awt.*;
import java.awt.event.*;

import javax.swing.*;
//import oracle.bali.ewt.border.UIBorderFactory;
//import oracle.bali.ewt.olaf.OracleLookAndFeel;

public class GUIStylesheet
{
  private static final boolean useBali = false;

  public GUIStylesheet()
  {
    Frame frame = new GUInterface();
    //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.setVisible(true);
  }

  public static void main(String[] args)
  {
    new GUIStylesheet();
  }
}
```

XML Process and Management Scripts

The XML process and management scripts used in the B2B XML application are as follows:

Java Example 6: Main4XMLtoDMLv2.java

```
package B2BDemo.XMLUtil;
/**
 * A main for tests
 * The XMLtoDMLv2 utility takes an XML document that can contain
 * data to be inserted in several tables.
 *
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
import java.io.*;
import java.net.*;

public class Main4XMLtoDMLv2 extends Object
{
  // Manage user input...
  private static BufferedReader _stdin = new BufferedReader(new
InputStreamReader(System.in));
  private static String _buf = "";

  private static String _userInput(String prompt) throws Exception
  {
    String retString;
    System.out.print(prompt);
    try { retString = _stdin.readLine(); }
```

```
    catch (Exception e)
    {
      System.out.println(e);
      throw(e);
    }
    return retString;
  }
  // for tests
  public static void main(String args[])
  {
    XMLtoDMLv2 x2d = new XMLtoDMLv2("scott", "tiger",

"jdbc:oracle:thin:@olediour-lap.us.oracle.com:1521:Ora8i");

    String xmldocname = "";
    try { xmldocname = userInput("XML file name > "); }
    catch (Exception e) {}
    String xmldoc = readURL(createURL(xmldocname));

    TableInDocument d[] = new TableInDocument[2];
    d[0] = new TableInDocument("ROWSET", "ROW", "DEPT");
    d[1] = new TableInDocument("EMP", "EMP_ROW", "EMP");

    try
    {
      x2d.insertFromXML(d, xmldoc);
      System.out.println(xmldocname + " processed.");
    }
    catch (Exception e)
    {
      System.err.println("Ooops:\n" + e);
    }

    try { _buf = _userInput("End of task..."); } catch (Exception ioe) {}
  }

  public 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.err.println("Cannot create url for: " + fileName);
      System.exit(0);
    }
  }
  return url;
}

public static String readURL(URL url)
{
  URLConnection newURLConn;
  BufferedInputStream newBuff;
  int nBytes;
  byte aByte[];
  String resultBuff = "";

  aByte = new byte[2];
  try
  {
//  System.out.println("Calling " + url.toString());
    try
    {
      newURLConn = url.openConnection();
      newBuff = new BufferedInputStream(newURLConn.getInputStream());
      resultBuff = "";
```

```
        while (( nBytes = newBuff.read(aByte, 0, 1)) != -1)
          resultBuff = resultBuff + (char)aByte[0];
      }
      catch (IOException e)
      {
        System.err.println(url.toString() + "\n : newURLConn failed :\n" + e);
      }
    }
    catch (Exception e) {}
    return resultBuff;
  }

  private static String userInput(String prompt) throws Exception
  {
    String retString;
    System.out.print(prompt);
    try { retString = _stdin.readLine(); }
    catch (Exception e)
    {
      System.out.println(e);
      throw(e);
    }
    return retString;
  }
}
```

Java Example 7: ParserTest.java

```
package B2BDemo.XMLUtil;

import org.xml.sax.*;
import java.io.*;
import java.util.*;
import java.net.*;
import java.sql.*;

import oracle.xml.sql.query.*;
import oracle.xml.sql.dml.*;

import org.w3c.dom.*;
import oracle.xml.parser.v2.*;
import org.xml.sax.*;
/**
```

```
 * Just a main for tests.
 * Show how to retrieve the ID and CUSTOMER_ID fro an XML document
 *
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
public class ParserTest extends Object
{

  static DOMParser parser = new DOMParser();

  static String XMLDoc =
"<ROWSET>" +
"   <ROW NUM=\"1\">" +
"      <ID>23321</ID>" +
"      <ORDERDATE>2000-05-03 00:00:00.0</ORDERDATE>" +
"      <CONTACTNAME>JDevBC4J</CONTACTNAME>" +
"      <TRACKINGNO>AX23321</TRACKINGNO>" +
"      <STATUS>Pending</STATUS>" +
"      <ITEMS>" +
"         <ITEM_ROW NUM=\"1\">" +
"            <ID>1242</ID>" +
"            <QUANTITY>2</QUANTITY>" +
"            <ITEM_ID>403</ITEM_ID>" +
"            <ORD_ID>23321</ORD_ID>" +
"            <DISCOUNT>0</DISCOUNT>" +
"         </ITEM_ROW>" +
"      </ITEMS>" +
"   </ROW>" +
"</ROWSET>";
  /**
   * Constructor
   */
  public ParserTest()
  {
  }

  public static void main(String[] args)
  {
    parser.setValidationMode(false);
    try
    {
      parser.parse(new InputSource(new
ByteArrayInputStream(XMLDoc.getBytes())));
      XMLDocument xml = parser.getDocument();
      XMLElement elmt = (XMLElement)xml.getDocumentElement();
```

```
      NodeList nl = elmt.getElementsByTagName("ID"); // ORD ID
      for (int i=0; i<nl.getLength(); i++)
      {
        XMLElement ordId = (XMLElement)nl.item(i);
        XMLNode theText = (XMLNode)ordId.getFirstChild();
        String ordIdValue = theText.getNodeValue();
        System.out.println(ordIdValue);
        break;
      }
      nl = elmt.getElementsByTagName("CUSTOMER_ID"); // CUSTOMER ID
      for (int i=0; i<nl.getLength(); i++)
      {
        XMLElement ordId = (XMLElement)nl.item(i);
        XMLNode theText = (XMLNode)ordId.getFirstChild();
        String custIdValue = theText.getNodeValue();
        System.out.println(custIdValue);
      }
    }
    catch (SAXParseException e)
    {
      System.out.println(e.getMessage());
    }
    catch (SAXException e)
    {
      System.out.println(e.getMessage());
    }
    catch (Exception e)
    {
      System.out.println(e.getMessage());
    }
  }
}
```

Java Example 8: TableInDocument.java

```
package B2BDemo.XMLUtil;
/**
 *   This class is used by the XMLtoDMLv2.java class
 *   It describes the matching between an XML document and a SQL table.
 *   Created to managed multi-level XML documents (Master-Details)
 *
 * @see XMLtoDMLv2
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
```

```
public class TableInDocument extends Object
{
  public String rowSet = "ROWSET";
  public String row    = "ROW";
  public String table  = "";

  public TableInDocument (String rset,
                          String r,
                          String t)
  {
    this.rowSet = rset;
    this.row    = r;
    this.table  = t;
  }
}
```

Java Example 9: XMLFrame.java

```
// Copyright (c) 2000 Oracle Corporation
package B2BDemo.XMLUtil;

import javax.swing.*;
import java.awt.*;
import oracle.xml.srcviewer.*;

import org.w3c.dom.*;
import oracle.xml.parser.v2.*;
import org.xml.sax.*;

/**
 * A Swing-based top level window class.
 * Implements the Code View of the Transviewer Bean.
 * Used in the demo to enhance the XML code propagated from one
 * component to another.
 *
 * @author Olivier LE DIOURIS
 */
public class XMLFrame extends JFrame
{
  BorderLayout borderLayout1 = new BorderLayout();
  JPanel jPanel1 = new JPanel();
  BorderLayout borderLayout2 = new BorderLayout();
  XMLSourceView xmlSourceViewPanel = new XMLSourceView();
```

```
  private String frameTitle = "";
  private XSLTWrapper xsltw = new XSLTWrapper();
  /**
   * Constructs a new instance.
   */
  public XMLFrame(String fTitle)
  {
    super();
    this.frameTitle = fTitle;
    try
    {
      jbInit();
    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
  }

  /**
   * Initializes the state of this instance.
   */
  private void jbInit() throws Exception
  {
    this.getContentPane().setLayout(borderLayout1);
    this.setSize(new Dimension(400, 300));
    jPanel1.setLayout(borderLayout2);
    this.setTitle(this.frameTitle);
    this.getContentPane().add(jPanel1, BorderLayout.CENTER);
    jPanel1.add(xmlSourceViewPanel, BorderLayout.CENTER);
  }

  public void setXMLDocument(String xmlContent) throws Exception
  {
    xmlSourceViewPanel.setXMLDocument(xsltw.parseDocument(xmlContent));
  }
}
```

Java Example 10: XMLProducer.java

```
package B2BDemo.XMLUtil;
/**
 * A Wrapper around the XML SQL Utility
 * Could be called from any java object
```

```
 * to produce an XML document after a SQL query,
 * not only from a servlet.
 *
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
/**
 */
import java.sql.*;
import oracle.xml.sql.query.*;

public class XMLProducer
{
  Connection conn = null;
  String rowset = null;
  String row = null;

  public XMLProducer(Connection conn)
  {
    this.conn = conn;
  }

  public String getXMLString(ResultSet rSet)
  {
    return getXMLString(rSet, "N");
  }

  public String getXMLString(ResultSet rSet,
                             String DTD)
  {
    String finalDoc = "";

    try
    {
      boolean dtdRequired = false;
      if (DTD != null && DTD.length() > 0 && DTD.toUpperCase().equals("Y"))
        dtdRequired = true;
      // The Skill ! ////////////////////////////////////////////
      OracleXMLQuery oXmlq = new OracleXMLQuery(conn, rSet);      //
  //  oXmlq.useUpperCaseTagNames();                               //
      if (this.rowset != null)
        oXmlq.setRowsetTag(this.rowset);
      if (this.row != null)
        oXmlq.setRowTag(this.row);
      finalDoc = oXmlq.getXMLString(dtdRequired);                 //
      // That's it ! ////////////////////////////////////////////
```

```
    }
    catch (Exception e)
    {
      System.err.println(e);
    }
    return finalDoc;
  }

  public void setRowset(String rSet)
  {
    this.rowset = rSet;
  }
  public void setRow(String row)
  {
    this.row = row;
  }
}
```

Java Example 11: XMLtoDMLv2.java

```
package B2BDemo.XMLUtil;
/**
 *    This class takes an XML document as input to execute
 *    an insert into the database.
 *    Multi level XML documents are supported, but not if
 *    one element has several sons as
 *      <elem1>
 *        <elem11/>
 *        <elem12/>
 *      </elem1>
 *
 * @see TableInDocument
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
import org.xml.sax.*;
import java.io.*;
import java.util.*;
import java.net.*;
import java.sql.*;

import oracle.xml.sql.query.*;
import oracle.xml.sql.dml.*;

import org.w3c.dom.*;
```

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

public class XMLtoDMLv2 extends Object
{
  static DOMParser parser = new DOMParser();
  Connection conn = null;
  String username = "";
  String password = "";
  String connURL = "";

  public XMLtoDMLv2(String username,
                    String password,
                    String connURL)
  {
    this.username = username;
    this.password = password;
    this.connURL  = connURL;
  }

  public void insertFromXML(TableInDocument tInDoc[],
                            String document) throws Exception
  {
    if (conn == null)
      getConnected();

    String xmlString = "";
    try
    { xmlString = readURL(createURL(document)); }
    catch (Exception e)
    { xmlString = document; }

//  System.out.println("xml2Insert = \n" + xmlString);

    // The returned String is turned into an XML Document
    XMLDocument xmlDoc = parseDocument(xmlString);
    // And we take a reference on the root of this document
    XMLElement e = (XMLElement) xmlDoc.getDocumentElement();

    // Let's walk thru the ROW nodes
    NodeList nl  = e.getChildrenByTagName(tInDoc[0].row); // "ROW"
//  System.out.println("This document has " + nl.getLength() + " ROW(s)");

    Vector sqlStmt = new Vector();
    scanLevel(0, tInDoc, nl, sqlStmt);
```

```
    // Now execute all the statements in the Vector, in reverse order (FK...)
    int i = sqlStmt.size();
    Enumeration enum = sqlStmt.elements();
    while (i > 0)
    {
      i--;
      String s = (String)sqlStmt.elementAt(i);
   // System.out.println("Executing " + s);
      executeStatement(s);
    }
  }

  // This one is recursive
  private int scanLevel(int level,
                        TableInDocument tInDoc[],
                        NodeList nl,
                        Vector sqlStmt) throws Exception
  {
    int nbRowProcessed = 0;
    Vector columnNames = new Vector();
    Vector columnValues = null;
    String[] colTypes = null;

    String columns = "", values = "";
    // Loop in tree...
    boolean firstLoop = true;
    for (int i=0; i<nl.getLength(); i++)  // Loop on all rows of XML doc
    {
      columnValues = new Vector();
      XMLElement aRow = (XMLElement) nl.item(i);
  //  String numVal   = aRow.getAttribute("num");
  //  System.out.println("NUM = " + numVal);
      NodeList nlRow = aRow.getChildNodes();
  //  System.out.println("a Row has " + nlRow.getLength() + " children");
      for (int j=0; j<nlRow.getLength(); j++)
      {
        XMLElement anXMLElement = (XMLElement)nlRow.item(j);
        if (anXMLElement.getChildNodes().getLength() == 1 &&
            (level == (tInDoc.length - 1) || (level < (tInDoc.length - 1) &&
!(anXMLElement.getNodeName().equals(tInDoc[level+1].rowSet)))) )
        {
      //  System.out.println("Element " + (j+1) + "=" +
anXMLElement.getNodeName());
      //  System.out.print(anXMLElement.getNodeName());
```

```
          if (firstLoop)
            columnNames.addElement(anXMLElement.getNodeName());
          // Value
          XMLNode nodeValue = (XMLNode) anXMLElement.getFirstChild();
      //  System.out.println("\t" + nodeValue.getNodeValue());
          columnValues.addElement(nodeValue.getNodeValue());
        }
        else
        {
   //     System.out.println(anXMLElement.getNodeName() + " has " +
anXMLElement.getChildNodes().getLength() + " children");
   //    System.out.println("Comparing " + anXMLElement.getNodeName() + " and "
+ tInDoc[level+1].rowSet);
          if (level < (tInDoc.length - 1) &&
anXMLElement.getNodeName().equals(tInDoc[level+1].rowSet))
          {
   //       System.out.println("Searching for " + tInDoc[level+1].row);
            NodeList nl2  =
anXMLElement.getChildrenByTagName(tInDoc[level+1].row); // "ROW"
            if (nl2 == null)
              System.out.println("Nl2 is null for " + tInDoc[level+1].row);
            scanLevel(level + 1, tInDoc, nl2, sqlStmt);
          }
        }
      }
 //   System.out.println("INSERT INTO " + tableName + " (" + columns + ") VALUES
(" + values + ")");
      try
      {
        if (firstLoop)
        {
          firstLoop = false;
          String selectStmt = "SELECT ";
          boolean comma = false;
          Enumeration cNames = columnNames.elements();
          while (cNames.hasMoreElements())
          {
            columns += ((comma?", ":"") + (String)cNames.nextElement());
            if (!comma)
              comma = true;
          }
          selectStmt += columns;
         selectStmt += (" FROM " + tInDoc[level].table + " WHERE 1 = 2"); // No
row retrieved
          Statement stmt = conn.createStatement();
```

```
   //     System.out.println("Executing: " + selectStmt);
          ResultSet rSet = stmt.executeQuery(selectStmt);
          ResultSetMetaData rsmd = rSet.getMetaData();
          colTypes = new String[rsmd.getColumnCount()];
          for (int ci=0; ci<(rsmd.getColumnCount()); ci++)
          {
        //  System.out.println("Col " + (ci+1) + ":" + rsmd.getColumnName(ci+1)
+ ", " + rsmd.getColumnTypeName(ci+1));
            colTypes[ci] = rsmd.getColumnTypeName(ci+1);
          }
          rSet.close();
          stmt.close();
        }
        // Build Value Part
        int vi = 0;
        Enumeration cVal = columnValues.elements();
        boolean comma = false;
        while (cVal.hasMoreElements())
        {
          if (comma)
            values += ", ";
          comma = true;
          if (colTypes[vi].equals("DATE"))
            values += ("TO_DATE(SUBSTR(");
          values += ("'" + cVal.nextElement() + "'");
          if (colTypes[vi].equals("DATE"))
            values += (", 1, 19), 'YYYY-MM-DD HH24:MI:SS')");
          vi++;
        }
        // Go !
    //  System.out.println("Stmt:" + "INSERT INTO " + tInDoc[level].table + " ("
+ columns + ") VALUES (" + values + ")");
        sqlStmt.addElement("INSERT INTO " + tInDoc[level].table + " (" + columns
+ ") VALUES (" + values + ")");
        nbRowProcessed++;
      }
      catch (Exception execE)
      {
 //     System.err.println("Executing " + execE);
        throw execE;
      }
      values = "";
    }
//  System.out.println("End of Loop");
    return nbRowProcessed;
```

```
  }

  public static XMLDocument parseDocument(String documentStream) throws
Exception
  {
    XMLDocument returnXML = null;
    try
    {
      parser.parse(new InputSource(new
ByteArrayInputStream(documentStream.getBytes())));
      returnXML = parser.getDocument();
    }
    catch (SAXException saxE)
    {
 //   System.err.println("Parsing XML\n" + "SAX Exception:\n" +
saxE.toString());
 //   System.err.println("For:\n" + documentStream + "\nParse failed SAX : " +
saxE);
      throw saxE;
    }
    catch (IOException e)
    {
 //   System.err.println("Parsing XML\n" + "Exception:\n" + e.toString());
 //   System.err.println("Parse failed : " + e);
      throw e;
    }
    return returnXML;
  }
  // Create a URL from a file name
  private static URL createURL(String fileName) throws Exception
  {
    URL url = null;
    try
    {
      url = new URL(fileName);
    }
    catch (MalformedURLException ex) // It is not a valid URL, maybe a file...
    {
      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.err.println("Cannot create url for: " + fileName);
      throw e;   // It's not a file either...
    }
  }
  return url;
}

private static String readURL(URL url) throws Exception
{
  URLConnection newURLConn;
  BufferedInputStream newBuff;
  int nBytes;
  byte aByte[];
  String resultBuff = "";

  aByte = new byte[2];
  try
  {
 // System.out.println("Calling " + url.toString());
    try
    {
      newURLConn = url.openConnection();
      newBuff = new BufferedInputStream(newURLConn.getInputStream());
      resultBuff = "";
      while (( nBytes = newBuff.read(aByte, 0, 1)) != -1)
        resultBuff = resultBuff + (char)aByte[0];
    }
    catch (IOException e)
    {
 //   System.err.println("Opening locator\n" + e.toString());
 //   System.err.println(url.toString() + "\n : newURLConn failed :\n" + e);
```

```
      throw e;
    }
  }
  catch (Exception e)
  {
 // System.err.println("Read URL\n" + e.toString());
    throw e;
  }
  return resultBuff;
}

private void executeStatement(String strStmt) throws SQLException, Exception
{
  if (conn == null)
    throw new Exception("Connection is null");
  try
  {
    Statement stmt = conn.createStatement();
    stmt.execute(strStmt);
    stmt.close();
  }
  catch (SQLException e)
  {
    System.err.println("Failed to execute statement\n" + strStmt);
    throw e;
  }
}

private void getConnected() throws Exception
{
  try
  {
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    conn = DriverManager.getConnection(connURL, username, password);
  }
  catch (Exception e)
  {
//   System.err.println(e);
    throw e;
  }
}
public Connection getConnection()
{
  return conn;
}
```

```
}
```

Java Example 12: XMLGen.java

```
package B2BDemo.XMLUtil;

import java.sql.*;
/**
 * This class is used by the Action Handler called by the XSQL Servlet
 * in placeOrder.xsql. It generates the original XML Document to be
 * sent to the broker
 *
 * @see B2BMessage
 * @see XMLProducer
 * @see RetailActionHandler
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
public class XMLGen extends Object
{
  static Connection conn = null;
  // Default connection parameters
  static String appURL      = "jdbc:oracle:thin:@localhost:1521:ORCL";
  static String appUser     = "retailer";
  static String appPassword = "retailer";

  static String XMLStmt =
  "SELECT O.ID as \"Id\","               +
  "       O.ORDERDATE as \"Orderdate\", " +
  "       O.CONTACTNAME as \"Contactname\"," +
  "       O.TRACKINGNO as \"Trackingno\"," +
  "       O.STATUS as \"Status\"," +
  "       O.CUSTOMER_ID as \"CustomerId\"," +
  "       CURSOR (SELECT L.ID as \"Id\"," +
  "                      L.QUANTITY as \"Quantity\", " +
  "                      L.ITEM_ID as \"ItemId\"," +
  "                      L.ORD_ID as \"OrdId\"," +
  "                      L.DISCOUNT as \"Discount\" " +
  "               FROM LINE_ITEM L " +
  "               WHERE L.ORD_ID = O.ID) as \"LineItemView\" " +
  "FROM ORD O " +
  "WHERE O.ID = ?";

  public static String returnDocument (Connection c, String ordId)
  {
```

```
  String XMLDoc = "";
  try
  {
    if (c != null)
      conn = c;
    if (conn == null)
      _getConnected(appURL, appUser, appPassword);
    XMLProducer xmlp = null;
    xmlp = new XMLProducer(conn);  // The XML Producer
    xmlp.setRowset("Results");
    xmlp.setRow("OrdView");
    PreparedStatement stmt = conn.prepareStatement(XMLStmt);
    stmt.setString(1, ordId);
    ResultSet rSet = stmt.executeQuery();

    XMLDoc = xmlp.getXMLString(rSet, "Y");
    rSet.close();
    stmt.close();
    if (c == null)
    {
      conn.close();
      conn = null;
    }
  }
  catch (SQLException e)
  {}
  return XMLDoc;
}

private static void _getConnected(String connURL,
                                  String userName,
                                  String password)
{
  try
  {
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    conn = DriverManager.getConnection(connURL, userName, password);
  }
  catch (Exception e)
  {
    System.err.println(e);
    System.exit(1);
  }
}
```

```
  public static void main (String[] args) // Just for test !!
  {
    System.out.println(returnDocument(null, "28004"));
  }
}
```

Java Example 13: XMLUtil.java

```
package B2BDemo.XMLUtil;
/**
 * Matches a record of the Stylesheet table in the AQ Schema.
 *
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
public class XslRecord
{
  public String FROM;
  public String TO;
  public String TASK;
  public String XSL;

  public XslRecord(String FROM,
                   String TO,
                   String TASK,
                   String XSL)
  {
    this.FROM = FROM;
    this.TO = TO;
    this.TASK = TASK;
    this.XSL = XSL;
  }

  public boolean equals(XslRecord x)
  {
    if (this.FROM.equals(x.FROM) &&
        this.XSL.equals(x.XSL) &&
        this.TASK.equals(x.TASK) &&
        this.TO.equals(x.TO))
      return true;
    else
      return false;
  }
}
```

Java Example 14: XSLTWrapper.java

```
package B2BDemo.XMLUtil;
/**
 * Wraps some parser features.
 *
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
import java.net.*;
import java.io.*;

import org.w3c.dom.*;
import oracle.xml.parser.v2.*;
import org.xml.sax.*;

/**
 * This class is a wrapper for the XSL Transformer provided with the
 * Oracle XML Parser for Java V2.
 *
 * It processes XSL Transformations from Strings, files or URL as well.
 *
 * @author Olivier Le Diouris. Partner Services. Oracle Corp.
 * @version 1.0
 *
 */
public class XSLTWrapper
{
  DOMParser parser;

  String xml    = "";
  String xsl    = "";
  String result = "";

  private static boolean _debug = false;

  public XSLTWrapper()
  {
    parser = new DOMParser();
  }

  public void process() throws Exception
  {
    if (xml.length() == 0)
      throw new Exception("XML Document is empty");
    if (xsl.length() == 0)
```

```
      throw new Exception("XSL Document is empty");
    result = processTransformation(xml, xsl);
  }

  public void putXml(String xml) throws Exception
  {
    if (_debug) System.out.println("Recieved XML : \n" + xml);
    this.xml = xml;
  }
  public void putXsl(String xsl) throws Exception
  {
    this.xsl = xsl;
    if (_debug) System.out.println("Recieved XSL: \n" + xsl);
  }
  public String getXml() throws Exception
  {
    return xml;
  }
  public String getXsl() throws Exception
  {
    return xsl;
  }
  public String getProcessResult() throws Exception
  {
    return result;
  }

  // Turns a String into an XMLDocument
  public XMLDocument parseDocument(String documentStream) throws Exception
  {
    XMLDocument returnXML = null;
    try
    {
      parser.parse(new InputSource(new
ByteArrayInputStream(documentStream.getBytes())));
      returnXML = parser.getDocument();
    }
    catch (SAXException saxE)
    {
     if (_debug) System.err.println("For:\n" + documentStream + "\nParse failed
SAX : " + saxE);
      throw new Exception("Parsing XML\n" + "SAX Exception:\n" +
saxE.toString());
    }
    catch (IOException e)
```

```
  {
    if (_debug) System.err.println("Parse failed : " + e);
    throw new Exception("Parsing XML\n" + "IOException:\n" + e.toString());
  }
  return returnXML;
}

private XMLDocument processXML(XMLDocument xml,
                               XMLDocument xslDoc) throws Exception
{
  XMLDocument out = null;
  URL xslURL = null;

  try
  {
    parser.setPreserveWhitespace(true);
    parser.setValidationMode(false);       // Validation. Should be an option.
    // instantiate a stylesheet
    XSLStylesheet xsl = new XSLStylesheet(xslDoc, xslURL);
    XSLProcessor processor = new XSLProcessor();

    // display any warnings that may occur
    processor.showWarnings(true);
    processor.setErrorStream(System.err);

    // Process XSL
    DocumentFragment result = processor.processXSL(xsl, xml);

    // create an output document to hold the result
    out = new XMLDocument();
    /*
    // create a dummy document element for the output document
    Element root = out.createElement("root");
    out.appendChild(root);
    // append the transformed tree to the dummy document element
    root.appendChild(result);
     */
    out.appendChild(result);
    // print the transformed document
    // out.print(System.out);
  }
  catch (Exception e)
  {
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    PrintWriter pw = new PrintWriter(baos);
```

```
    e.printStackTrace(pw);
    e.printStackTrace();
    throw new Exception("ProcessXML\n" +  baos.toString());
  }
  return(out);
}

/**
 *   XML String and XSL String as input
 *     Input Strings may content :
 *            the name of a URL
 *            the name of a file (on the local file system)
 *            the document itself
 *   XML String as output.
 */
public String processTransformation(String xmlStream,
                                    String xslStream) throws Exception
{
  String xmlContent = "";
  String xslContent = "";

  try
  { xmlContent = readURL(createURL(xmlStream)); }
  catch (Exception e)
  { xmlContent = xmlStream; }

  try
  { xslContent = readURL(createURL(xslStream)); }
  catch (Exception e)
  { xslContent = xslStream; }

  if (_debug) System.out.println("xmlStream = " + xmlContent);
  if (_debug) System.out.println("xslStream = " + xslContent);

  XMLDocument xml = parseDocument(xmlContent);
  XMLDocument xsl = parseDocument(xslContent);

  XMLDocument out = processXML(xml, xsl);
  ByteArrayOutputStream baos = new ByteArrayOutputStream();
  try
  { out.print(baos); }
  catch (IOException ioE)
  {
    if (_debug) System.err.println("Exception:" + ioE);
    throw new Exception("XML Processing throws IOException\n" +
```

```
ioE.toString());
    }
    return (baos.toString());
  }

  // Create a URL from a file name
  private static URL createURL(String fileName) throws Exception
  {
    URL url = null;
    try
    {
      url = new URL(fileName);
    }
    catch (MalformedURLException ex) // It is not a valid URL, maybe a file...
    {
      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)
      {
        if (_debug) System.err.println("Cannot create url for: " + fileName);
        throw e;   // It's not a file either...
      }
    }
    return url;
  }

  private static String readURL(URL url) throws Exception
  {
```

```
    URLConnection newURLConn;
    BufferedInputStream newBuff;
    int nBytes;
    byte aByte[];
    String resultBuff = "";

    aByte = new byte[2];
    try
    {
   // System.out.println("Calling " + url.toString());
      try
      {
        newURLConn = url.openConnection();
        newBuff = new BufferedInputStream(newURLConn.getInputStream());
        resultBuff = "";
        while (( nBytes = newBuff.read(aByte, 0, 1)) != -1)
          resultBuff = resultBuff + (char)aByte[0];
      }
      catch (IOException e)
      {
   //   System.err.println("Opening locator\n" + e.toString());
   //   System.err.println(url.toString() + "\n : newURLConn failed :\n" + e);
        throw e;
      }
    }
    catch (Exception e)
    {
   // System.err.println("Read URL\n" + e.toString());
      throw e;
    }
    return resultBuff;
  }
}
```

Other Scripts Used in the B2B XML Application

XML Example 1: XSQL Configuration — XSQLConfig.xml

```
<?xml version="1.0" ?>
<!--
| $Author: smuench $
| $Date: 2000/03/14 10:36:42 $
```

```
| $Source: C:\\cvsroot/xsql/src/XSQLConfig.xml,v $
| $Revision: 1.11 $
+-->
<XSQLConfig>

  <!--
  |
  | This section defines configuration settings
  | specific to the XSQL Servlet
  |
  +-->
  <servlet>

   <!--
    |
    | 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.
    |
    |    <output-buffer-size>10000</output-buffer-size>
    |
    +-->
    <output-buffer-size>0</output-buffer-size>

   <!--
    |
    | Add <media-type> elements as shown below to cause
    | the XSQL Servlet to *suppress* sending the "charset=XXX"
    | portion of the Media/Content-type.
    |
    | For example, sending a character set for "image/svg"
    | documents seems to confuse current SVG plugins.
    |
    | <suppress-mime-charset>
    |   <media-type>image/svg</media-type>
    | </suppress-mime-charset>
    |
    +-->

    <suppress-mime-charset>
       <media-type>image/svg</media-type>
    </suppress-mime-charset>

  </servlet>
```

```
<!--
 |
 | This section defines XSQL Page Processor configuration settings.
 |
 +-->
 <processor>

   <!--
   |
   | Connection definitions (see <connectiondefs> below)
   | are cached when the XSQL Page Processor is initialized.
   |
   | Set 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.
   |
   +-->

   <reload-connections-on-error>yes</reload-connections-on-error>

   <!--
   |
   | Set the default value of the Row Fetch Size
   | for retrieving information from SQL queries
   | from the database. Only takes effect if you
   | are using the Oracle JDBC Driver, otherwise
   | the setting is ignored. Useful for reducing
   | network roundtrips to the database from
   | the servlet engine running in a different tier.
   |
   |     <default-fetch-size>50</default-fetch-size>
   |
   +-->

   <default-fetch-size>50</default-fetch-size>

   <!--
```

```
|
| Set the value of the XSQL LRU Cache for XSQL Pages
| This determines the maximum number of stylesheets
| that will be cached. Least recently used sheets get
| "bumped" out of the cache if you go beyond this number.
|
| <page-cache-size>25</page-cache-size>
|
+-->

<page-cache-size>25</page-cache-size>

<!--
|
| Set the value of the XSQL LRU Cache for XSL Stylesheets.
| This determines the maximum number of stylesheets
| that will be cached. Least recently used sheets get
| "bumped" out of the cache if you go beyond this number.
|
|     <stylesheet-cache-size>25</stylesheet-cache-size>
|
+-->

<stylesheet-cache-size>25</stylesheet-cache-size>

<!--
|
| Set the parameters controlling stylesheet pools.
|
| Each cached stylesheet is actually a cached pool
| of stylesheet instances. These values control
| The initial number of stylesheet instances in the
| pool, the number that will be added/incremented
| when under-load the pool must be grown, and
| the number of seconds that must transpire without
| activity before a stylesheet instance will be
| dropped out of the pool to shrink it back towards
| its initial number.
|
| <stylesheet-pool>
|   <initial>1</initial>
|   <increment>1</increment>
|   <timeout-seconds>60</timeout-seconds>
| </stylesheet-pool>
|
```

```
+-->

<stylesheet-pool>
  <initial>1</initial>
  <increment>1</increment>
  <timeout-seconds>60</timeout-seconds>
</stylesheet-pool>

<!--
|
| Set the parameters controlling database connection pools.
|
| When used, each named connection defined can have a pool of
| connection instances to share among requests.  These values
| control The initial number of stylesheet instances in the pool,
| the number that will be added/incremented when under-load the
| pool must be grown, and the number of seconds that must
| transpire without activity before a stylesheet instance will be
| dropped out of the pool to shrink it back towards its initial
| number.
|
| If the "dump-allowed" element has the value "yes"
| then a browser-based status report that dumps the
| current state of the connection pools is enabled.
|
| <connection-pool>
|   <initial>2</initial>
|   <increment>1</increment>
|   <timeout-seconds>60</timeout-seconds>
|   <dump-allowed>no</dump-allowed>
| </connection-pool>
|
+-->

<connection-pool>
  <initial>2</initial>
  <increment>1</increment>
  <timeout-seconds>60</timeout-seconds>
  <dump-allowed>no</dump-allowed>
</connection-pool>

<!--
|
| Include timing information (in Milliseconds)
|
```

```
    | <timing-info>
    |   <page>yes</page>
    |  <action>yes</action>
    | </timing-info>
    |
    +-->

    <timing-info>
      <page>no</page>
      <action>no</action>
    </timing-info>

  </processor>

  <!--
  |
  | This section defines HTTP Proxy Server name
  | and port for use by the <xsql:include-xml>
  | action. If you intend to use <xsql:include-xml>
  | to include XML from URL's outside a firewall,
  | uncomment the:
  |
  |  <http>
  |    <proxyhost>your-proxy-server.yourcompany.com</proxyhost>
  |    <proxyport>80</proxyport>
  |  </http>
  |
  | section below and change the proxyhost and proxyport
  | as appropriate. If left commented out, then the XSQL
  | Page processor does not use a proxy server.
  |
  +-->

<!--

  <http>
    <proxyhost>your-proxy-server.yourcompany.com</proxyhost>
    <proxyport>80</proxyport>
  </http>

-->

  <!--
  |
  | This section defines convenient "nicknames" for
```

```
| one or more database connections. You can include
| any number of <connection> elements inside of
| the <connectiondefs> element. XSQL Pages refer to
| these connections by their name in the "connection"
| attribute on the document element of the page.
|
+-->

<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="xmlbook">
    <username>xmlbook</username>
    <password>xmlbook</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>
  <connection name="retail">
    <username>retailer</username>
    <password>retailer</password>
    <dburl>jdbc:oracle:thin:@atp-1.us.oracle.com:1521:ORCL</dburl>
    <driver>oracle.jdbc.driver.OracleDriver</driver>
  </connection>

</connectiondefs>

<!--
|
| This section registers pre-defined element names and
| handler classes for user-defined XSQL page actions
|
| The section looks like:
|
| <actiondefs>
```

```
  |  <action>
  |    <elementname>myAction</elementname>
  |    <handlerclass>mypackage.MyActionHandler</handlerclass>
  |  </action>
  |      :
  | <actiondefs>
  |
  | Action Handler classes must implement the interface
  | oracle.xml.xsql.XSQLActionHandler.
  |
  | Once registered here, user-defined actions can be
  | used in the same way as built-in XSQL actions, for example
  | including the <xsql:myAction> element in your page.
  |
  +-->
  <actiondefs>
    <action>
      <elementname>param</elementname>

<handlerclass>oracle.xml.xsql.actions.ExampleGetParameterHandler</handlerclass>
    </action>
    <action>
      <elementname>current-date</elementname>

<handlerclass>oracle.xml.xsql.actions.ExampleCurrentDBDateHandler</handlerclass>
    </action>
  </actiondefs>

</XSQLConfig>
```

Java Example 15: Message Header Script — MessageHeaders.java

The message header script is given here:

```
package B2BDemo;
/**
 * Describes the headers used in the messages
 *
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
public class MessageHeaders extends Object
{
  public static String APP_A          = "RETAIL";
  public static String APP_B          = "SUPPLY";
```

```
  public static String BROKER         = "BROKER";
  public static String EXIT           = "EXIT";
  public static String NEW_ORDER      = "NEW ORDER";
  public static String UPDATE_ORDER   = "UPDATE ORDER";
}
```

Java Example 16: Hold Constants for Use by Message Broker — AppCste.java

```
package B2BDemo;
/**
 * Holds the constants to be used by the Message Broker
 *
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
public class AppCste extends Object
{
 public final static String AQDBUrl =
  "jdbc:oracle:thin:@atp-1.us.oracle.com:1521:ORCL";
  public final static String AQuser  = "aqMessBrok";
  public final static String AQpswd  = "aqMessBrok";
}
```

Retailer Scripts

The Retailer uses the following scripts:

- Java Example 17: Retailer Waits for Status Update Sent from Supplier — UpdateMaster.java

Java Example 17: Retailer Waits for Status Update Sent from Supplier — UpdateMaster.java

```
package B2BDemo.Retailer;
/**
 *
 * This class implements the component waiting on the retailer side for
 * the status update made by the Supplier after shipping.
 * The recieved document is parsed and its content is used to make
 * the convenient update in the database.
 *
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 *
 */
```

```
import java.io.*;
import java.util.*;
import java.net.*;
import java.sql.*;

import oracle.xml.sql.query.*;
import oracle.xml.sql.dml.*;

import org.w3c.dom.*;
import oracle.xml.parser.v2.*;
import org.xml.sax.*;

import B2BDemo.AQUtil.*;
import B2BDemo.*;
import B2BDemo.XMLUtil.*;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
//import oracle.bali.ewt.border.UIBorderFactory;
//import oracle.bali.ewt.olaf.OracleLookAndFeel;

public class UpdateMaster extends Object
{
  private BufferedReader stdin = new BufferedReader(new
InputStreamReader(System.in));

  private static boolean stepByStep = false;
  private static boolean verbose    = false;
  private static Integer pauseTime  = null;

  AQReader aqr;

  private static final String userName = "retailer";
  private static final String password = "retailer";
  private static String url      = "jdbc:oracle:thin:@localhost:1521:ORCL"; //
This is the default value !
  private static Connection conn = null;

  String currOrdId = "";

  DOMParser parser = new DOMParser();
  /**
   * Constructor
   */
```

```
  public UpdateMaster()
  {
    XMLFrame frame = new XMLFrame("Retailer");
    /**
    try
    {
      OracleLookAndFeel.setColorScheme(Color.cyan);
//    OracleLookAndFeel.setColorScheme("Titanium");
      UIManager.setLookAndFeel(new OracleLookAndFeel());
      SwingUtilities.updateComponentTreeUI(frame);
      frame.setBackground(UIManager.getColor("darkIntensity"));
    }
    catch (Exception e)
    {
      System.err.println("Exception for Oracle Look and Feel:" + e );
    }
    */
    //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;
    }
    */
    frameSize.width = screenSize.width / 3;

//  frame.setLocation((screenSize.width - frameSize.width)/2, (screenSize.height
- frameSize.height)/2);
    frame.setLocation(0, (screenSize.height - frameSize.height)/2);
//  frame.addWindowListener(new WindowAdapter() { public void
windowClosing(WindowEvent e) { System.exit(0); } });
    frame.setVisible(true);

      // Initialize AQ reader
    aqr = new AQReader(AppCste.AQuser,
                       AppCste.AQpswd,
                       AppCste.AQDBUrl,
                       "AppFour_QTab",
                       "AppFourMsgQueue");
```

```
    boolean go = true;
    while (go)
    {
      String ordIdValue = "";
      B2BMessage sm = aqr.readQ();
      if (verbose)
        System.out.println("Recieved\nFrom > " + sm.getFrom() +
                                   "\nTo   > " + sm.getTo() +
                                   "\nType > " + sm.getType() +
                                   "\nContent >\n" + sm.getContent());
      else
        System.out.println("Recieved\nFrom > " + sm.getFrom() +
                                   "\nTo   > " + sm.getTo() +
                                   "\nType > " + sm.getType());
      String xmlDoc = sm.getContent();
      if (xmlDoc != null && xmlDoc.length() > 0)
      {
        try { frame.setXMLDocument(sm.getContent()); }
        catch (Exception e)
        {
          e.printStackTrace();
        }
      }
      if (stepByStep)
      {
        if (pauseTime != null)
        {
          System.out.println("Waiting for " + pauseTime.longValue() + "
milliseconds");
          try { Thread.sleep(pauseTime.longValue()); } catch
(InterruptedException e) {}
        }
        else
          try { String s = _userInput("[Hit return to continue]"); } catch
(Exception e) {}
      }

      if (sm.getType().equals(MessageHeaders.EXIT))
        go = false;
      else
      {
        System.out.println("Updating");
        try
        {
          parser.parse(new InputSource(new
```

```
ByteArrayInputStream(sm.getContent().getBytes()));
          XMLDocument xml = parser.getDocument();
          XMLElement elmt = (XMLElement)xml.getDocumentElement();
          NodeList nl = elmt.getElementsByTagName("SHIP"); // ORD ID
          for (int i=0; i<nl.getLength(); i++)
          {
            XMLElement ordId = (XMLElement)nl.item(i);
            XMLNode theText = (XMLNode)ordId.getFirstChild();
            currOrdId = theText.getNodeValue();
            System.out.println("Gonna update " + currOrdId);
            try
            {
              if (conn == null)
                getConnected(url, userName, password);
              String strStmt = "update ORD set STATUS = 'Shipped' where ID = ?";
              PreparedStatement pStmt = conn.prepareStatement(strStmt);
              pStmt.setString(1, currOrdId);
              pStmt.execute();
              conn.commit();
              pStmt.close();
              System.out.println("Done !");
            }
            catch (SQLException e)
            {
              System.out.println("Pb updating the ORD\n" + e.toString());
            }

          }
        }
        catch (SAXParseException e)
        {
          System.out.println(e.getMessage());
        }
        catch (SAXException e)
        {
          System.out.println(e.getMessage());
        }
        catch (Exception e)
        {
          System.out.println(e.getMessage());
        }
      }
    }
    frame.setVisible(false);
    System.exit(0);
```

```
  }

  private static void getConnected(String connURL,
                                   String userName,
                                   String password)
  {
    try
    {
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
      conn = DriverManager.getConnection(connURL, userName, password);
    }
    catch (Exception e)
    {
      System.err.println(e);
      System.exit(1);
    }
  }

  private String _userInput(String prompt) throws Exception
  {
    String retString;
    System.out.print(prompt);
    try { retString = stdin.readLine(); }
    catch (Exception e)
    {
      System.out.println(e);
      throw(e);
    }
    return retString;
  }

  private static void setRunPrm(String[] prm)
  {
    for (int i=0; i<prm.length; i++)
    {
      if (prm[i].toLowerCase().startsWith("-verbose"))
        verbose = isolatePrmValue(prm[i], "-verbose");
      else if (prm[i].toLowerCase().startsWith("-help"))
      {
        System.out.println("Usage is:");
        System.out.println("\tjava B2BDemo.Retailer.MessageBroker");
        System.out.println("\tparameters can be -dbURL -verbose, -step, -help");
        System.out.println("\tdbURL contains a string like
jdbc:oracle:thin:@localhost:1521:ORCL");
        System.out.println("\tparameters values can be (except for -help):");
```

```
      System.out.println("\t\tnone - equivalent to 'y'");
      System.out.println("\t\ty");
      System.out.println("\t\ttrue - equivalent to 'y'");
      System.out.println("\t\tn");
      System.out.println("\t\tfalse - equivalent to 'n'");
      System.out.println("\t\t-step can take a value in milliseconds");
      System.exit(0);
    }
    else if (prm[i].toLowerCase().startsWith("-step"))
    {
      String s = getPrmValue(prm[i], "-step");
      try
      {
        pauseTime = new Integer(s);
        System.out.println("Timeout " + pauseTime);
        stepByStep = true;
      }
      catch (NumberFormatException nfe)
      {
        pauseTime = null;
        if (s.toUpperCase().equals("Y") || s.toUpperCase().equals("TRUE"))
          stepByStep = true;
        else
          stepByStep = false;
      }
    }
    else if (prm[i].toLowerCase().startsWith("-dburl"))
    {
      url = getPrmValue(prm[i], "-dbURL");
    }
    else
      System.err.println("Unknown parameter [" + prm[i] + "], ignored.");
  }
}

private static boolean isolatePrmValue(String s, String p)
{
  boolean ret = true;
  if (s.length() > (p.length() + 1)) // +1 : "="
  {
    if (s.indexOf("=") > -1)
    {
      String val = s.substring(s.indexOf("=") + 1);
      if (val.toUpperCase().equals("Y") || val.toUpperCase().equals("TRUE"))
        ret = true;
```

```
        else if (val.toUpperCase().equals("N") ||
val.toUpperCase().equals("FALSE"))
          ret = false;
        else
        {
          System.err.println("Unrecognized value for " + p + ", set to y");
          ret = true;
        }
      }
    }
    return ret;
  }

  private static String getPrmValue(String s, String p)
  {
    String ret = "";
    if (s.length() > (p.length() + 1)) // +1 : "="
    {
      if (s.indexOf("=") > -1)
      {
        ret = s.substring(s.indexOf("=") + 1);
      }
    }
    return ret;
  }

  /**
   * main
   * @param args
   */
  public static void main(String[] args)
  {
    if (args.length > 0)
      setRunPrm(args);
    UpdateMaster updateMaster = new UpdateMaster();
  }
}
```

AQ Broker-Transformer and Advanced Queuing Scripts

The AQ-Broker-Transformer uses the following scripts:

- Java Example 18: AQ Broker Listens on One AQ Thread — BrokerThread.java

Java Example 18: AQ Broker Listens on One AQ Thread — BrokerThread.java

```
package B2BDemo.Broker;

import java.sql.*;
import oracle.AQ.*;
import java.io.*;
import oracle.sql.*;
import oracle.jdbc.driver.*;

import B2BDemo.AQUtil.*;
import B2BDemo.XMLUtil.*;
import B2BDemo.*;

/**
 * This class implements a thread listening on one AQ.
 *
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
public class BrokerThread extends Thread
{
  private BufferedReader stdin = new BufferedReader(new
InputStreamReader(System.in));

  private static boolean stepByStep = false;
  private static boolean verbose    = false;

  AQReader aqReader;
  AQWriter aqWriter;
  String threadName;
  XSLTWrapper wrapper;
  Connection conn;
  Integer pause;
```

```
  XMLFrame frame;
  /**
   * Constructor
   */
  public BrokerThread(String name,
                      AQReader aqr,
                      AQWriter aqw,
                      XSLTWrapper wrap,
                      Connection c,
                      boolean v,
                      boolean s,
                      Integer p,
                      XMLFrame f)
  {
    this.aqReader = aqr;
    this.aqWriter = aqw;
    this.threadName = name;
    this.wrapper = wrap;
    this.conn = c;

    this.verbose = v;
    this.stepByStep = s;
    this.pause = p;
    this.frame = f;
  }

  public void run()
  {
    boolean go = true;
    while (go)
    {
      B2BMessage sm = this.aqReader.readQ();
      if (verbose)
        System.out.println(this.threadName + " Recieved\nFrom > " + sm.getFrom()
+
                                    "\nTo   > " + sm.getTo() +
                                    "\nType > " + sm.getType() +
                                    "\nContent >\n" + sm.getContent());
      else
        System.out.println(this.threadName + " Recieved\nFrom > " + sm.getFrom()
+
                                    "\nTo   > " + sm.getTo() +
                                    "\nType > " + sm.getType());
      String xmlDoc = sm.getContent();
      if (xmlDoc != null && xmlDoc.length() > 0)
```

```
{
  try { this.frame.setXMLDocument(sm.getContent()); }
  catch (Exception e)
  {
    e.printStackTrace();
  }
}
if (stepByStep)
{
  if (pause != null)
  {
    System.out.println("Waiting for " + pause.longValue() + "
                                                        milliseconds");
    try { Thread.sleep(pause.longValue()); } catch (InterruptedException
                                                              e) {}
  }
  else
    try { String s = _userInput("[Hit return to continue]"); } catch
                                                    (Exception e) {}
}
if (sm.getType().length() >= MessageHeaders.EXIT.length() &&
                              sm.getType().equals(MessageHeaders.EXIT))
  go = false;
else
{
  // Transform !
  String processedXMLDoc = "";
  String xslDoc = getXSL(sm.getFrom(),
                         sm.getTo(),
                         sm.getType());
  if (verbose)
    System.out.println("Read:\n" + xslDoc);
  try
  {
    processedXMLDoc = wrapper.processTransformation(sm.getContent(),
                                       xslDoc /*defaultStyleSheet*/);
    if (verbose)
      System.out.println("\nResult :\n" + processedXMLDoc);
    System.out.println("Transformation done.");
  }
  catch (Exception e)
  {
    System.err.println("Ooops...\n");
    e.printStackTrace();
  }
```

```
        if (stepByStep)
        {
          if (pause != null)
          {
            System.out.println("Waiting for " + pause.longValue() + "
                                                              milliseconds");
            try { Thread.sleep(pause.longValue()); } catch (InterruptedException
                                                              e) {}
          }
          else
            try { String s = _userInput("[Hit return to continue]"); } catch
                                                              (Exception e) {}
        }

        // Send new document to destination
        this.aqWriter.writeQ(new B2BMessage(sm.getFrom(),
                                            sm.getTo(),
                                            sm.getType(),
                                            processedXMLDoc));
        this.aqWriter.flushQ();
      }
    }
    if (frame.isVisible())
      frame.setVisible(false);
    System.exit(0);
  }

  private String getXSL(String from,
                        String to,
                        String task)
  {
    if (verbose)
      System.out.println("Processing From " + from + " to " + to + " for " +
task);
    String xsl = "";
    String stmt = "SELECT XSL FROM STYLESHEETS WHERE APPFROM = ? AND APPTO = ?
AND OP = ?";

    try
    {
      PreparedStatement pStmt = conn.prepareStatement(stmt);
      pStmt.setString(1, from);
      pStmt.setString(2, to);
      pStmt.setString(3, task);
      ResultSet rSet = pStmt.executeQuery();
```

```
    while (rSet.next())
      xsl = _dumpClob(conn, ((OracleResultSet)rSet).getCLOB(1));
    rSet.close();
    pStmt.close();
  }
  catch (SQLException e)
  { }
  catch (Exception e)
  { }
  return xsl;
}

static String _dumpClob (Connection conn, CLOB clob)
  throws Exception
{
  String returnStr = "";

  OracleCallableStatement cStmt1 =
    (OracleCallableStatement)
      conn.prepareCall ("begin ? := dbms_lob.getLength (?); end;");
  OracleCallableStatement cStmt2 =
    (OracleCallableStatement)
      conn.prepareCall ("begin dbms_lob.read (?, ?, ?, ?); end;");

  cStmt1.registerOutParameter (1, Types.NUMERIC);
  cStmt1.setClob (2, clob);
  cStmt1.execute ();

  long length = cStmt1.getLong (1);
  long i = 0;
  int chunk = 100;
  if (verbose)
    System.out.println("Length to read from DB : " + length);

  while (i < length)
  {
    cStmt2.setClob (1, clob);
    cStmt2.setLong (2, chunk);
    cStmt2.registerOutParameter (2, Types.NUMERIC);
    cStmt2.setLong (3, i + 1);
    cStmt2.registerOutParameter (4, Types.VARCHAR);
    cStmt2.execute ();

    long readThisTime = cStmt2.getLong (2);
    String stringThisTime = cStmt2.getString (4);
```

```
//    System.out.print ("Read " + read_this_time + " chars: ");
      returnStr += stringThisTime;
      i += readThisTime;
    }

    cStmt1.close ();
    cStmt2.close ();

    return returnStr;
  }

  private String _userInput(String prompt) throws Exception
  {
    String retString;
    System.out.print(prompt);
    try { retString = stdin.readLine(); }
    catch (Exception e)
    {
      System.out.println(e);
      throw(e);
    }
    return retString;
  }
}
```

Java Example 19: MessageBroker.java

```
package B2BDemo.Broker;
/**
 * Implements the AQ Broker-Transformer.
 * This "Message Broker" uses 4 message queues, provided by
 * Oracle8i Advanced Queuing.
 * AQ Broker uses the threads described in BrokerThread
 * Each thread is waiting on one queue, and writing on another.
 * The message broker uses - for this demo - two threads :
 *   One from retailer to supplier
 *   One from supplier to retailer
 * 2 Threads := 4 queues
 *
 * When a message is recieved, the broker knows :
 *   where it comes from (origin)
 *   where it goes to (destination)
 *   what for (operation)
```

```
 * Those three elements are used as the primary key for the
 * stylesheet table (belonging to the AQ schema) to fetch the
 * right sXSL Stylesheet from the database, in order to turn
 * the incoming document into an outgoing one fitting the requirements
 * of the destination application.
 *
 * @see BrokerThread
 * @see B2BMessage
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
import java.sql.*;
import oracle.AQ.*;
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import oracle.sql.*;
import oracle.jdbc.driver.*;
//import oracle.bali.ewt.border.UIBorderFactory;
//import oracle.bali.ewt.olaf.OracleLookAndFeel;

import B2BDemo.AQUtil.*;
import B2BDemo.*;
import B2BDemo.XMLUtil.*;

public class MessageBroker extends Object
{
  private static boolean stepByStep = false;
  private static boolean verbose    = false;
  private static Integer pauseTime  = null;

  XSLTWrapper wrapper = null;

  // To get the style sheet from its CLOB
  Connection conn = null;
  String userName = AppCste.AQuser;
  String password = AppCste.AQpswd;
  String dbUrl    = AppCste.AQDBUrl;

  public MessageBroker()
  {
    XMLFrame frame = new XMLFrame("Message Broker");
    /**
    try
    {
```

```
      OracleLookAndFeel.setColorScheme(Color.cyan);
//    OracleLookAndFeel.setColorScheme("Titanium");
      UIManager.setLookAndFeel(new OracleLookAndFeel());
      SwingUtilities.updateComponentTreeUI(frame);
      frame.setBackground(UIManager.getColor("darkIntensity"));
    }
    catch (Exception e)
    {
      System.err.println("Exception for Oracle Look and Feel:" + e );
    }
    */
    //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;
    }
    */
    frameSize.width = screenSize.width / 3;
//  frame.setLocation((screenSize.width - frameSize.width)/2, (screenSize.height
- frameSize.height)/2);
    frame.setLocation(frameSize.width, (screenSize.height -
frameSize.height)/2);
//  frame.addWindowListener(new WindowAdapter() { public void
windowClosing(WindowEvent e) { System.exit(0); } });
    frame.setVisible(true);

    AQReader aqr = null;
    AQWriter aqw = null;
    // Initialize AQ reader and writer
    aqw = new AQWriter(AppCste.AQuser,
                       AppCste.AQpswd,
                       AppCste.AQDBUrl,
                       "AppTwo_QTab",
                       "AppTwoMsgQueue");
    aqr = new AQReader(AppCste.AQuser,
                       AppCste.AQpswd,
                       AppCste.AQDBUrl,
                       "AppOne_QTab",
```

```
                    "AppOneMsgQueue");
  wrapper = new XSLTWrapper();
  if (conn == null)
    _getConnected();

  BrokerThread retail2supply = new BrokerThread("Retail to Supply",
                                                aqr,
                                                aqw,
                                                wrapper,
                                                conn,
                                                verbose,
                                                stepByStep,
                                                pauseTime,
                                                frame);
  aqw = new AQWriter(AppCste.AQuser,
                     AppCste.AQpswd,
                     AppCste.AQDBUrl,
                     "AppFour_QTab",
                     "AppFourMsgQueue");
  aqr = new AQReader(AppCste.AQuser,
                     AppCste.AQpswd,
                     AppCste.AQDBUrl,
                     "AppThree_QTab",
                     "AppThreeMsgQueue");
  BrokerThread supply2retail = new BrokerThread("Supply to Retail",
                                                aqr,
                                                aqw,
                                                wrapper,
                                                conn,
                                                verbose,
                                                stepByStep,
                                                pauseTime,
                                                frame);

  retail2supply.start();
  supply2retail.start();

  System.out.println("<ThreadsOnTheirWay/>");
}

private void _getConnected()
{
  try
  {
    Class.forName ("oracle.jdbc.driver.OracleDriver");
```

```
    conn = DriverManager.getConnection (dbUrl, userName, password);
  }
  catch (Exception e)
  {
    System.out.println("Get connected failed : " + e);
    System.exit(1);
  }
}

private static void setRunPrm(String[] prm)
{
  for (int i=0; i<prm.length; i++)
  {
    if (prm[i].toLowerCase().startsWith("-verbose"))
      verbose = isolatePrmValue(prm[i], "-verbose");
    else if (prm[i].toLowerCase().startsWith("-help"))
    {
      System.out.println("Usage is:");
      System.out.println("\tjava Intel.iDevelop.MessageBroker");
      System.out.println("\tparameters can be -verbose, -step, -help");
      System.out.println("\tparameters values can be (except for -help):");
      System.out.println("\t\tnone - equivalent to 'y'");
      System.out.println("\t\ty");
      System.out.println("\t\ttrue - equivalent to 'y'");
      System.out.println("\t\tn");
      System.out.println("\t\tfalse - equivalent to 'n'");
      System.out.println("\t\t-step can take a value in milliseconds");
      System.exit(0);
    }
    else if (prm[i].toLowerCase().startsWith("-step"))
    {
      String s = getPrmValue(prm[i], "-step");
      try
      {
        pauseTime = new Integer(s);
        System.out.println("Timeout " + pauseTime);
        stepByStep = true;
      }
      catch (NumberFormatException nfe)
      {
        pauseTime = null;
        if (s.toUpperCase().equals("Y") || s.toUpperCase().equals("TRUE"))
          stepByStep = true;
        else
          stepByStep = false;
```

```
        }
      }
      else
        System.err.println("Unknown parameter [" + prm[i] + "], ignored.");
    }
  }

  private static boolean isolatePrmValue(String s, String p)
  {
    boolean ret = true;
    if (s.length() > (p.length() + 1)) // +1 : "="
    {
      if (s.indexOf("=") > -1)
      {
        String val = s.substring(s.indexOf("=") + 1);
        if (val.toUpperCase().equals("Y") || val.toUpperCase().equals("TRUE"))
          ret = true;
        else if (val.toUpperCase().equals("N") ||
val.toUpperCase().equals("FALSE"))
          ret = false;
        else
        {
          System.err.println("Unrecognized value for " + p + ", set to y");
          ret = true;
        }
      }
    }
    return ret;
  }

  private static String getPrmValue(String s, String p)
  {
    String ret = "";
    if (s.length() > (p.length() + 1)) // +1 : "="
    {
      if (s.indexOf("=") > -1)
      {
        ret = s.substring(s.indexOf("=") + 1);
      }
    }
    return ret;
  }

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

```
    // java B2BDemo.OrderEntry.MessageBroker -verbose[=[y|true|n|false]]
-step[=[y|true|n|false]] -help
    if (args.length > 0)
      setRunPrm(args);

    new MessageBroker();
  }
}
```

Java Example 20: AQReader.java

```
package B2BDemo.AQUtil;
/**
 * This class is a wrapper around the Advanced Queuing facility of Oracle 8i.
 * Used to dequeue a message.
 *
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
import java.sql.*;
import oracle.AQ.*;
import java.io.*;

public class AQReader extends Object
{
  Connection conn  = null;
  AQSession aqSess = null;

  String userName   = "";
  String qTableName = "";
  String qName      = "";

  AQQueueTable aqTable = null;
  AQQueue      aq      = null;

  public AQReader(String userName,
                  String password,
                  String url,
                  String qTable,
                  String qName)
  {
    this.userName = userName;
    this.qTableName = qTable;
    this.qName = qName;
    aqSess = createSession(userName, password, url);
```

```
  aqTable = aqSess.getQueueTable(userName, qTableName);
  System.out.println("Successful getQueueTable");
  // Handle to q
  aq = aqSess.getQueue(userName, qName);
  System.out.println("Successful getQueue");
}

public AQSession createSession(String userName,
                               String pswd,
                               String url)
{
  try
  {
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    conn = DriverManager.getConnection(url, userName, pswd);
    System.out.println("JDBC Connection opened");
    conn.setAutoCommit(false);
    // Load Oracle 8i AQ Driver
    Class.forName("oracle.AQ.AQOracleDriver");
    // Create the AQ Session
    aqSess = AQDriverManager.createAQSession(conn);
    System.out.println("AQ Session successfully created.");
  }
  catch (Exception e)
  {
    System.out.println("Exception : " + e);
    e.printStackTrace();
  }
  return aqSess;
}

public B2BMessage readQ() throws AQException
{
  AQMessage message;
  AQRawPayload rawPayload;

  // Read with REMOVE option
  AQDequeueOption dqOption = new AQDequeueOption();
  dqOption.setDequeueMode(AQDequeueOption.DEQUEUE_REMOVE);
  message = aq.dequeue(dqOption);

  System.out.println("Successfull dQueue");
  rawPayload = message.getRawPayload();
```

```
    try
    {
      conn.commit(); // Commit the REMOVE
    }
    catch (Exception sqle)
    {
      System.err.println(sqle.toString());
    }

    return (B2BMessage)deserializeFromByteArray(rawPayload.getBytes());
  }

  private static Object deserializeFromByteArray (byte[] b)
  {
    ByteArrayInputStream inputStream = new ByteArrayInputStream(b);
    try
    {
      ObjectInputStream ois = new ObjectInputStream(inputStream);
    return ois.readObject();
    }
    catch (Exception e)
    {
      System.err.println("deserializeFromByteArray failed :" + e);
      return null;
    }
  }
}
```

Java Example 21: AQWriter.java

```
package B2BDemo.AQUtil;
/**
 * This class is a wrapper around the Advanced Queuing facility of Oracle 8i.
 * Used to enqueue a message.
 *
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
import java.sql.*;
import oracle.AQ.*;
import java.io.*;

public class AQWriter extends Object
{
  Connection conn = null;
```

```
AQSession aqSess = null;

String userName   = "";
String qTableName = "";
String qName      = "";

public AQWriter(String userName,
                String password,
                String url,
                String qTable,
                String qName)
{
  this.userName = userName;
  this.qTableName = qTable;
  this.qName = qName;
  aqSess = createSession(userName, password, url);
}

public void flushQ()
{
  if (conn != null)
  {
    try { conn.commit(); } catch (SQLException e) {}
  }
}

public void closeConnection()
{
  if (conn != null)
  {
    try { conn.close(); }
    catch (SQLException e)
    { }
  }
}

public AQSession createSession(String userName,
                               String pswd,
                               String url)
{
  try
  {
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    conn = DriverManager.getConnection(url, userName, pswd);
    System.out.println("JDBC Connection opened");
```

```
    conn.setAutoCommit(false);
    // Load Oracle 8i AQ Driver
    Class.forName("oracle.AQ.AQOracleDriver");
    // Create the AQ Session
    aqSess = AQDriverManager.createAQSession(conn);
    System.out.println("AQ Session successfully created.");
  }
  catch (Exception e)
  {
    System.out.println("Exception : " + e);
    e.printStackTrace();
  }
  return aqSess;
}

public void writeQ(B2BMessage sm) throws AQException
{
  AQQueueTable         qTable;
  AQQueue              q;

  qTable = aqSess.getQueueTable(userName, qTableName);
  System.out.println("Successful getQueueTable");
  // Handle to q
  q = aqSess.getQueue(userName, qName);
  System.out.println("Successful getQueue");

  // Q is identified, let's write
  AQMessage message;
  AQRawPayload rawPayload;

  message = q.createMessage();
  byte[] bArray = serializeToByteArray(sm);
  rawPayload = message.getRawPayload();
  rawPayload.setStream(bArray, bArray.length);
  AQEnqueueOption eqOption = new AQEnqueueOption();
  q.enqueue(eqOption, message);
}

private static byte[] serializeToByteArray (Object o)
{
  ByteArrayOutputStream outStream = new ByteArrayOutputStream();
  try
  {
    ObjectOutputStream oos = new ObjectOutputStream(outStream);
    oos.writeObject(o);
```

```
      return outStream.toByteArray();
    }
    catch (Exception e)
    {
      System.err.println("serialize2ByteArray failed : " + e);
      return null;
    }
  }
}
```

Java Example 22: B2BMessage.java

```
package B2BDemo.AQUtil;
/**
 * This class decsribes the structure of the messages used in this demo
 * Subject to changes in 817
 *
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
import java.io.Serializable;

public class B2BMessage extends Object implements Serializable
{
  String from;
  String to;
  String type;
  String content;

  public B2BMessage(String f,
                    String t,
                    String typ,
                    String c)
  {
    this.from    = f;
    this.to      = t;
    this.type    = typ;
    this.content = c;
  }

  public String getFrom()
  { return this.from; }
  public String getTo()
  { return this.to; }
  public String getType()
```

```
  { return this.type; }
  public String getContent()
  { return this.content; }

}
```

Java Example 23: ReadStructAQ.java

```
package B2BDemo.AQUtil;
/**
 * A main for tests - Not used in the demo itself.
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
import java.sql.*;
import oracle.AQ.*;
import java.io.*;

import B2BDemo.*;

public class ReadStructAQ extends Object
{
  public static void main(String[] args)
  {
    AQReader aqr = new AQReader(AppCste.AQuser,
                                AppCste.AQpswd,
                                AppCste.AQDBUrl,
                                "objMsgsStruct_QTab",
                                "structMsgQueue");
    // Loop while EXIT is not recieved
    boolean goLoop = true;
    while (goLoop)
    {
      B2BMessage sm = aqr.readQ();
      System.out.println("Recieved\nFrom > " + sm.getFrom() +
                                "\nTo   > " + sm.getTo() +
                                "\nType > " + sm.getType() +
                                "\nContent >\n" + sm.getContent());
      if (sm.getType().equals("EXIT"))
        goLoop = false;
    }
    System.out.println("<bye/>");
  }
}
```

Java Example 24: StopAllQueues.java

```
package B2BDemo.AQUtil;

import B2BDemo.*;

/**
 * Used in the demo to stop the queues and the applications waiting on them.
 *
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
public class StopAllQueues extends Object
{
  /**
   * Constructor
   */
  public StopAllQueues()
  {
     AQWriter aqw1 = new AQWriter(AppCste.AQuser,
                                  AppCste.AQpswd,
                                  AppCste.AQDBUrl,
                                  "AppOne_QTab",
                                  "AppOneMsgQueue");
     aqw1.writeQ(new B2BMessage(MessageHeaders.APP_B,
                                MessageHeaders.APP_A,
                                MessageHeaders.EXIT,
                                ""));
     aqw1.flushQ();
     AQWriter aqw2 = new AQWriter(AppCste.AQuser,
                                  AppCste.AQpswd,
                                  AppCste.AQDBUrl,
                                  "AppTwo_QTab",
                                  "AppTwoMsgQueue");
     aqw2.writeQ(new B2BMessage(MessageHeaders.APP_B,
                                MessageHeaders.APP_A,
                                MessageHeaders.EXIT,
                                ""));
     aqw2.flushQ();
     AQWriter aqw3 = new AQWriter(AppCste.AQuser,
                                  AppCste.AQpswd,
                                  AppCste.AQDBUrl,
                                  "AppThree_QTab",
                                  "AppThreeMsgQueue");
     aqw3.writeQ(new B2BMessage(MessageHeaders.APP_B,
                                MessageHeaders.APP_A,
```

```
                                MessageHeaders.EXIT,
                                ""));
    aqw3.flushQ();
    AQWriter aqw4 = new AQWriter(AppCste.AQuser,
                                 AppCste.AQpswd,
                                 AppCste.AQDBUrl,
                                 "AppFour_QTab",
                                 "AppFourMsgQueue");
    aqw4.writeQ(new B2BMessage(MessageHeaders.APP_B,
                                MessageHeaders.APP_A,
                                MessageHeaders.EXIT,
                                ""));
    aqw4.flushQ();
  }

  /**
   * main
   * @param args
   */
  public static void main(String[] args)
  {
    StopAllQueues stopAllQueues = new StopAllQueues();
  }
}
```

Java Example 25: WriteStructAQ.java

```
package B2BDemo.AQUtil;
/**
 * A Main for tests - Not used in the demo itself.
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
/**
 * A main for tests
 */
import java.sql.*;
import oracle.AQ.*;
import java.io.*;

import B2BDemo.*;

public class WriteStructAQ extends Object
{
```

```
  private static BufferedReader stdin = new BufferedReader(new
InputStreamReader(System.in));


  static AQWriter aqw = null;

  public static void main(String[] args)
  {
    try
    {
      aqw = new AQWriter(AppCste.AQuser,
                         AppCste.AQpswd,
                         AppCste.AQDBUrl,
                         "objMsgsStruct_QTab",
                         "structMsgQueue");
      String messSubject     = "";
      String messTxt         = "";
      String messOrigin      = "";
      String messDestination = "";
      try
      {
        messOrigin      = userInput("Message Origin      > ");
        messDestination = userInput("Message Destination > ");
        messSubject     = userInput("Message Subject     > ");
        messTxt         = userInput("Message Text        > ");
      } catch (Exception e){}

      // Write the queue
      B2BMessage sm = new B2BMessage(messOrigin,
                                     messDestination,
                                     messSubject,
                                     messTxt);
      aqw.writeQ(sm);
      try { String s = userInput("Written"); }
      catch (Exception ne) {}
      aqw.closeConnection();
      try { String s = userInput("Closed !"); }
      catch (Exception ne) {}
    }
    catch (Exception e)
    {
      System.err.println("Arghh : " + e);
      e.printStackTrace();
      try { String s = userInput("..."); }
      catch (Exception ne) {}
```

```
    }
  }

  private static String userInput(String prompt) throws Exception
  {
    String retString;
    System.out.print(prompt);
    try { retString = stdin.readLine(); }
    catch (Exception e)
    {
      System.out.println(e);
      throw(e);
    }
    return retString;
  }
}
```

Supplier Scripts

The Supplier uses the following scripts:

- Java Example 26: SupplierFrame.java
- Java Example 27: Agent Wakes Up with Order Received from Retailer — SupplierWatcher.java

Java Example 26: SupplierFrame.java

```
package B2BDemo.Supplier;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

import java.io.*;
import java.util.*;
import java.net.*;
import java.sql.*;

import oracle.xml.sql.query.*;
import oracle.xml.sql.dml.*;

import org.w3c.dom.*;
import oracle.xml.parser.v2.*;
import org.xml.sax.*;

import B2BDemo.AQUtil.*;
import B2BDemo.*;
import B2BDemo.XMLUtil.*;

/**
 * This class implements the Frame suggesting to ship the order.'
 *
 * @see SupplierWatcher in the same package.
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
public class SupplierFrame extends JFrame
{
  private BufferedReader stdin = new BufferedReader(new
InputStreamReader(System.in));
  private static boolean stepByStep = false;
  private static boolean verbose    = false;
```

```
private static Integer pause      = null;
private XMLFrame frame            = null;

AQReader aqr;
XMLtoDMLv2 x2d = null;

String userName = "supplier";
String password = "supplier";
String url      = null;

String currOrdId = "";

DOMParser parser = new DOMParser();

AQWriter aqw = null;

BorderLayout borderLayout1 = new BorderLayout();
JPanel jPanel1 = new JPanel();
BorderLayout borderLayout2 = new BorderLayout();
JPanel southPanel = new JPanel();
JButton shipButton = new JButton();
JPanel centerPanel = new JPanel();
JLabel ordMessage = new JLabel();

/**
 * Constructs a new instance.
 */
public SupplierFrame(boolean v, boolean s, Integer p, XMLFrame f, String url)
{
  super();
  this.verbose = v;
  this.stepByStep = s;
  this.pause = p;
  this.frame = f;
  this.url = url;
  try
  {
    jbInit();
  }
  catch (Exception e)
  {
    e.printStackTrace();
  }
}
```

```
/**
 * Initializes the state of this instance.
 */
private void jbInit() throws Exception
{
  this.getContentPane().setLayout(borderLayout1);
  this.setSize(new Dimension(400, 300));
  shipButton.setText("Ship Order");
  shipButton.setEnabled(false);
  shipButton.addActionListener(new java.awt.event.ActionListener()
  {

    public void actionPerformed(ActionEvent e)
    {
      shipButton_actionPerformed(e);
    }
  });
  ordMessage.setText("Waiting for Orders");
  ordMessage.setFont(new Font("Dialog", 1, 20));
  jPanel1.setLayout(borderLayout2);
  this.setTitle("Supplier Watcher");
  this.getContentPane().add(jPanel1, BorderLayout.CENTER);
  jPanel1.add(southPanel, BorderLayout.SOUTH);
  southPanel.add(shipButton, null);
  jPanel1.add(centerPanel, BorderLayout.CENTER);
  centerPanel.add(ordMessage, null);
}

public void enterTheLoop()
{
    // Initialize AQ reader
  aqr = new AQReader(AppCste.AQuser,
                     AppCste.AQpswd,
                     AppCste.AQDBUrl,
                     "AppTwo_QTab",
                     "AppTwoMsgQueue");
  // Initialize XSL Transformer
  x2d = new XMLtoDMLv2(userName,
                       password,
                       url);
  // Initialize the AQ Writer
  aqw = new AQWriter(AppCste.AQuser,
                     AppCste.AQpswd,
                     AppCste.AQDBUrl,
                     "AppThree_QTab",
```

```
                        "AppThreeMsgQueue");

    boolean go = true;
    while (go)
    {
      String ordIdValue = "";
      String custIdValue = "";
      B2BMessage sm = aqr.readQ();
      if (verbose)
        System.out.println("Recieved\nFrom > " + sm.getFrom() +
                                  "\nTo   > " + sm.getTo() +
                                  "\nType > " + sm.getType() +
                                  "\nContent >\n" + sm.getContent());
      else
        System.out.println("Recieved\nFrom > " + sm.getFrom() +
                                  "\nTo   > " + sm.getTo() +
                                  "\nType > " + sm.getType());
      String xmlDoc = sm.getContent();
      if (xmlDoc != null && xmlDoc.length() > 0)
      {
        try { this.frame.setXMLDocument(sm.getContent()); }
        catch (Exception e)
        {
          e.printStackTrace();
        }
      }
      if (stepByStep)
      {
        if (pause != null)
        {
          System.out.println("Waiting for " + pause.longValue() + "
milliseconds");
          try { Thread.sleep(pause.longValue()); } catch (InterruptedException
e) {}
        }
        else
          try { String s = _userInput("[Hit return to continue]"); } catch
(Exception e) {}
      }
      if (sm.getType().equals(MessageHeaders.EXIT))
        go = false;
      else
      {
        System.out.println("Inserting");
        TableInDocument d[] = new TableInDocument[2];
```

```
            d[0] = new TableInDocument("ROWSET", "ROW", "ORD");
            d[1] = new TableInDocument("ITEMS", "ITEM_ROW", "LINE_ITEM");
            try
            {
              String XMLDoc = sm.getContent();
              x2d.insertFromXML(d, XMLDoc);
              System.out.println("Document processed.");
              // We want to read elements
              parser.setValidationMode(false);
              try
              {
                parser.parse(new InputSource(new
ByteArrayInputStream(XMLDoc.getBytes())));
                XMLDocument xml = parser.getDocument();
                XMLElement elmt = (XMLElement)xml.getDocumentElement();
                NodeList nl = elmt.getElementsByTagName("ID"); // ORD ID
                for (int i=0; i<nl.getLength(); i++)
                {
                  XMLElement ordId = (XMLElement)nl.item(i);
                  XMLNode theText = (XMLNode)ordId.getFirstChild();
                  ordIdValue = theText.getNodeValue();
                  currOrdId = ordIdValue;
                  break; // Just the first one !!!
                }
                nl = elmt.getElementsByTagName("CUSTOMER_ID"); // CUSTOMER ID
                for (int i=0; i<nl.getLength(); i++)
                {
                  XMLElement ordId = (XMLElement)nl.item(i);
                  XMLNode theText = (XMLNode)ordId.getFirstChild();
                  custIdValue = theText.getNodeValue();
                }
              }
              catch (SAXParseException e)
              {
                System.out.println(e.getMessage());
              }
              catch (SAXException e)
              {
                System.out.println(e.getMessage());
              }
              catch (Exception e)
              {
                System.out.println(e.getMessage());
              }
            }
```

```
        catch (Exception e)
        {
          System.err.println("Ooops:\n" + e);
        }
        this.shipButton.setEnabled(true);
        String custName = "";
        try
        {
         PreparedStatement pStmt = x2d.getConnection().prepareStatement("Select
C.NAME from CUSTOMER C where C.ID = ?");
          pStmt.setString(1, custIdValue);
          ResultSet rSet = pStmt.executeQuery();
          while (rSet.next())
            custName = rSet.getString(1);
          rSet.close();
          pStmt.close();
        }
        catch (SQLException e)
        {}

        this.ordMessage.setText("Order [" + ordIdValue + "] to process for [" +
custName + "]");
        JOptionPane.showMessageDialog(this, "New Order Pending !", "Wake Up !",
                                      JOptionPane.INFORMATION_MESSAGE);
      }
    }
    frame.setVisible(false);
  }

  void shipButton_actionPerformed(ActionEvent e)
  {
    // Send message
    String doc2send = "<SHIP>" + currOrdId + "</SHIP>";
    // sending XMLDoc in the Queue
    aqw.writeQ(new B2BMessage(MessageHeaders.APP_B,
                              MessageHeaders.APP_A,
                              MessageHeaders.UPDATE_ORDER,
                              doc2send));
    aqw.flushQ();  // Commit !

    // Disable Button
    this.shipButton.setEnabled(false);
    // display wait message
    this.ordMessage.setText("Waiting for orders...");
  }
```

```
  private String _userInput(String prompt) throws Exception
  {
    String retString;
    System.out.print(prompt);
    try { retString = stdin.readLine(); }
    catch (Exception e)
    {
      System.out.println(e);
      throw(e);
    }
    return retString;
  }
}
```

Java Example 27: Agent Wakes Up with Order Received from Retailer — SupplierWatcher.java

```
package B2BDemo.Supplier;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
//import oracle.bali.ewt.border.UIBorderFactory;
//import oracle.bali.ewt.olaf.OracleLookAndFeel;

import B2BDemo.XMLUtil.*;

/**
 * This class implements the agent waiting on the queue where the orders are
delivered.
 * When a message is read, the agent "wakes up" and suggests to ship the order.
 * Shipping the order will then fire a new B2B process to update the status of
 * the order in the Retailer database.
 *
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
public class SupplierWatcher
{
  private static boolean stepByStep = false;
  private static boolean verbose    = false;
```

```
  private static Integer pauseTime  = null;
  private static String url      = "jdbc:oracle:thin:@localhost:1521:ORCL";  //
Default value
  /**
   * Constructor
   */
  public SupplierWatcher()
  {
    XMLFrame xmlFrame = new XMLFrame("Supplier");
    /*
    try
    {
      OracleLookAndFeel.setColorScheme(Color.cyan);
//    OracleLookAndFeel.setColorScheme("Titanium");
      UIManager.setLookAndFeel(new OracleLookAndFeel());
      SwingUtilities.updateComponentTreeUI(xmlFrame);
      xmlFrame.setBackground(UIManager.getColor("darkIntensity"));
    }
    catch (Exception e)
    {
      System.err.println("Exception for Oracle Look and Feel:" + e );
    }
    */
    //Center the window
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = xmlFrame.getSize();
    if (frameSize.height > screenSize.height)
    {
      frameSize.height = screenSize.height;
    }
    /**
    if (frameSize.width > screenSize.width)
    {
      frameSize.width = screenSize.width;
    }
    */
    frameSize.width = screenSize.width / 3;
//  xmlFrame.setLocation((screenSize.width - frameSize.width)/2,
(screenSize.height - frameSize.height)/2);
    xmlFrame.setLocation((2 * frameSize.width), (screenSize.height -
frameSize.height)/2);
//  xmlFrame.addWindowListener(new WindowAdapter() { public void
windowClosing(WindowEvent e) { System.exit(0); } });
    xmlFrame.setVisible(true);
```

```
    SupplierFrame frame = new SupplierFrame(verbose, stepByStep, pauseTime,
xmlFrame, url);
    /*
    try
    {
      OracleLookAndFeel.setColorScheme(Color.cyan);
//    OracleLookAndFeel.setColorScheme("Titanium");
      UIManager.setLookAndFeel(new OracleLookAndFeel());
      SwingUtilities.updateComponentTreeUI(frame);
      frame.setBackground(UIManager.getColor("darkIntensity"));
    }
    catch (Exception e)
    {
      System.err.println("Exception for Oracle Look and Feel:" + e );
    }
    */
    //Center the window
    screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    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);

    frame.enterTheLoop();
    frame.setVisible(false);
    xmlFrame.setVisible(false);
    System.exit(1);
  }

  private static void setRunPrm(String[] prm)
  {
    for (int i=0; i<prm.length; i++)
    {
      if (prm[i].toLowerCase().startsWith("-verbose"))
        verbose = isolatePrmValue(prm[i], "-verbose");
```

```
      else if (prm[i].toLowerCase().startsWith("-help"))
      {
        System.out.println("Usage iB2BDemo.Supplier.MessageBroker");
        System.out.println("\tparameters can be -dbURL -verbose, -step, -help");
        System.out.println("\tdbURL contains a string like
jdbc:oracle:thin:@localhost:1521:ORCL");
        System.out.println("\tparameters values can be (except for -help):");
        System.out.println("\t\tnone - equivalent to 'y'");
        System.out.println("\t\ty");
        System.out.println("\t\ttrue - equivalent to 'y'");
        System.out.println("\t\tn");
        System.out.println("\t\tfalse - equivalent to 'n'");
        System.out.println("\t\t-step can take a value in milliseconds");
        System.exit(0);
      }
      else if (prm[i].toLowerCase().startsWith("-step"))
      {
        String s = getPrmValue(prm[i], "-step");
        try
        {
          pauseTime = new Integer(s);
          System.out.println("Timeout " + pauseTime);
          stepByStep = true;
        }
        catch (NumberFormatException nfe)
        {
          pauseTime = null;
          if (s.toUpperCase().equals("Y") || s.toUpperCase().equals("TRUE"))
            stepByStep = true;
          else
            stepByStep = false;
        }
      }
      else if (prm[i].toLowerCase().startsWith("-dburl"))
      {
        url = getPrmValue(prm[i], "-dbURL");
      }
      else
        System.err.println("Unknown parameter [" + prm[i] + "], ignored.");
    }
  }

  private static boolean isolatePrmValue(String s, String p)
  {
    boolean ret = true;
```

```
    if (s.length() > (p.length() + 1)) // +1 : "="
    {
      if (s.indexOf("=") > -1)
      {
        String val = s.substring(s.indexOf("=") + 1);
        if (val.toUpperCase().equals("Y") || val.toUpperCase().equals("TRUE"))
          ret = true;
        else if (val.toUpperCase().equals("N") ||
val.toUpperCase().equals("FALSE"))
          ret = false;
        else
        {
          System.err.println("Unrecognized value for " + p + ", set to y");
          ret = true;
        }
      }
    }
    return ret;
  }

  private static String getPrmValue(String s, String p)
  {
    String ret = "";
    if (s.length() > (p.length() + 1)) // +1 : "="
    {
      if (s.indexOf("=") > -1)
      {
        ret = s.substring(s.indexOf("=") + 1);
      }
    }
    return ret;
  }

  /**
   * main
   * @param args
   */
  public static void main(String[] args)
  {
    if (args.length > 0)
      setRunPrm(args);

    try
    {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
```

```
    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
    new SupplierWatcher();
  }
}
```

14

Using JDeveloper to Build Oracle XML Applications

This chapter contains the following sections:

Introducing JDeveloper 3.2

JDeveloper Version 3.2, offers an integrated, full-featured application development tool for building and deploying applications in Java and XML.

You can use JDeveloper to build, debug and deploy Internet applications that create and process XML data and documents.

Oracle JDeveloper 3.2 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, whererby developers can edit and debug Oracle XSQL Pages, Java programs that can query the database and return formatted XML, or insert XML into the database without writing code. The integrated servlet engine allows you to view XML output generated by Java code in the same environment as your program source, making it easy to do rapid, iterative development and testing.
- Includes Oracle's XML Parser for Java
- Includes the XSL-T Processor
- Related JavaBeans components
- XSQL Page Wizard. See "Page Selector Wizard" on page 14-9.
- XSQL Element Wizard. See "XSQL Element Wizard" on page 14-7.
- XSQL ActionHandlers

Business Components for Java (BC4J)

Oracle Business Components for Java is a 100%-Java, XML-powered framework that enables productive development, portable deployment, and flexible customization of multi-tier, database-savvy applications from reusable business components.

Application developers use the Oracle Business Components framework and Oracle JDeveloper 's integrated design-time wizards, component editors, and productive Java coding environment to assemble and test application services from reusable business components.

These application services can then be deployed as either CORBA Server Objects or EJB Session Beans on enterprise-scale server platforms supporting Java technology.

The same server-side business component can be deployed without modification as either a JavaServer Pages / Servlet application or Enterprise JavaBeans component. This deployment flexibility, enables developers to reuse the same business logic and data models to deliver applications to a variety of clients, browsers, and wireless Internet devices without having to rewrite code.

In JDeveloper, you can customize the functionality of existing Business Components by using the new visual wizards to modify your XML metadata descriptions.

Oracle JDeveloper XML Strategy

This chapter steps you through developing a mobile application using the XML components in JDeveloper. It also provides a roadmap for JDeveloper's XML support. It describes the XML support currently available in JDeveloper as well as JDeveloper's future direction.

What's Needed to Run JDeveloper 3.2

JDeveloper 3.2 runs on Windows NT version 4.0 with a minimum of 128 Mb RAM.

Minimum system requirements for JDeveloper

Refer to JDeveloper Release Notes. As more products are run run on the same machine, system requirements are increased. A typical development environment for running JDeveloper includes:

- Running JDeveloper
- Running Oracle8*i* locally
- Running (Internet Application Server) *i*AS locally
- Additional third party tools (profilers, version control, modelers,...)

These add to system requirements, in terms of actual CPU usage and in disk space needs.

Accessing JDeveloper 3.2

JDeveloper 3.2 can be ordered or you can download JDeveloper 3.2 from the Oracle technology Network (OTN) at http://technet.oracle.com.

XML in Business Components for Java (BC4J)

The Business Components for Java (BC4J) framework in JDeveloper 3.2 uses XML to define the metadata that represents the declarative settings and features of the objects. Custom or complex business logic can be implemented in Java.

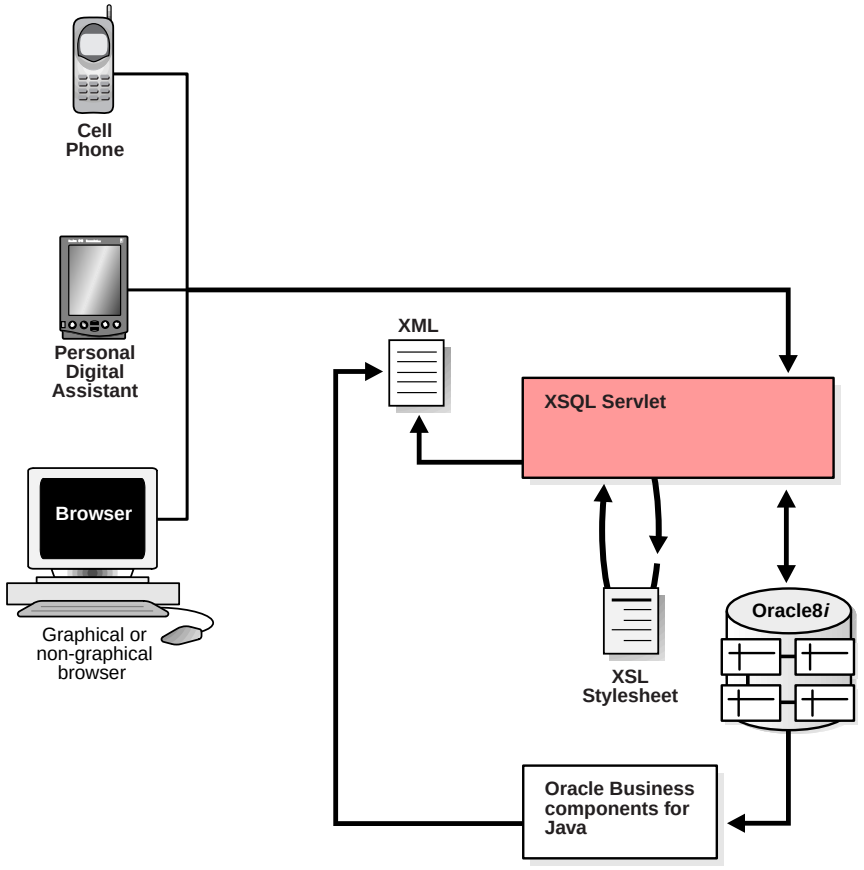
- The BC4J Tester enables you to see data in view objects as XML.
- Business rules, such as validation rules, are stored in XML rather than Java source code
- Easy customization of business applications by changing XML rather than Java source code
- Applications are easier to read and understand by abstracting the logic in XML

BC4J uses XML to Store Metadata The business components for Java framework that ships with JDeveloper uses XML to store metadata about its application components. Important information is now stored in a structured document rather than in Java source code. This makes the application easier to understand and customize.

The application is now customizable without having access to the source code.

Figure 14–1 shows how BC4J is used with XSQL servlet to generate XML documents.

Figure 14–1 Using Business Components for Java (BC4J)



Business rules can be changed on site without needing access to the underlying component source code.

Building XSQL Clients with Business Components for Java (BC4J)

In JDeveloper 3.2, you can build XSQL Pages which can integrate with BC4J application modules and thereby serve application logic from the middle tier to multiple clients. You can retrieve XML data and present it to any kind of a client device just by applying the corresponding stylesheet.

The following features will assist you in building XSQL clients with BC4J:

- Object Gallery
- XSQL Element Wizard
- Page Selector Wizard

Note: These appearance of these features may differ in the JDeveloper 3.2 production version.

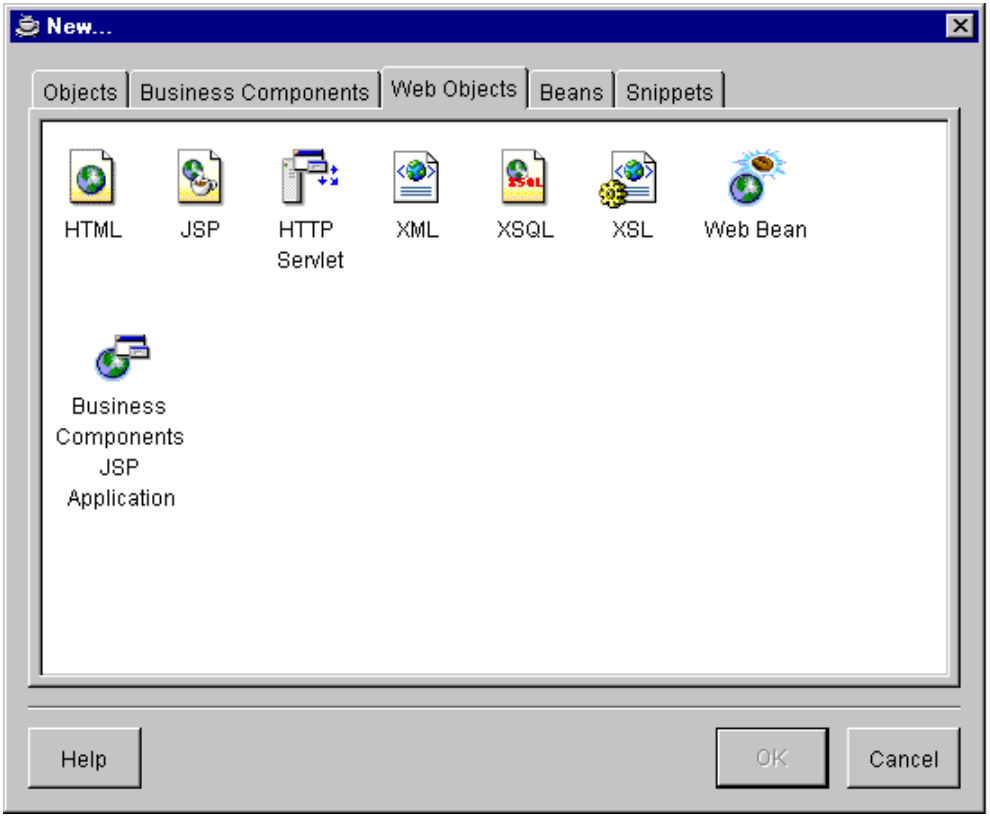
Object Gallery

The Web Object Gallery has three new icons to assist in creating XSQL, XML and XSL documents easily. When the user clicks on them, the basic tags for these pages are generated and the user can then enhance them.

The XSQL Pages icon is of special interest because XSQL Element Wizard can be used, after generating your basic XSQL pages, to insert data bound tags in the XSQL pages.

Figure 14–2 illustrates the Object Gallery.

Figure 14–2 JDeveloper's Object Gallery Showing the new XSQL, XML, and XSL Icons

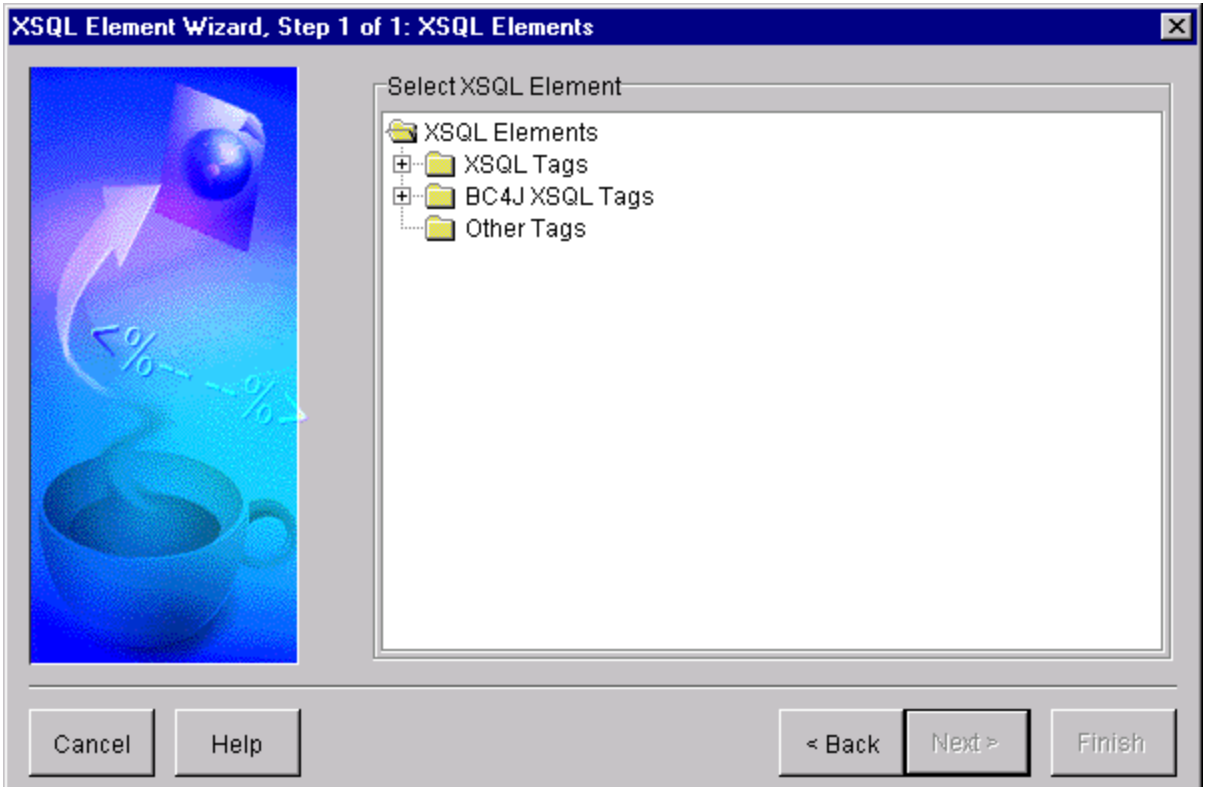


XSQL Element Wizard

XSQL Element Wizard provides you with a mechanism to add tags which allows accessing database tables or BC4J View Objects. You can either perform queries against them or update the underlying database tables through them.

Figure 14–3 illustrates the JDeveloper 3.2 XSQL Element Wizard.

Figure 14–3 JDeveloper's XSQL Element Wizard

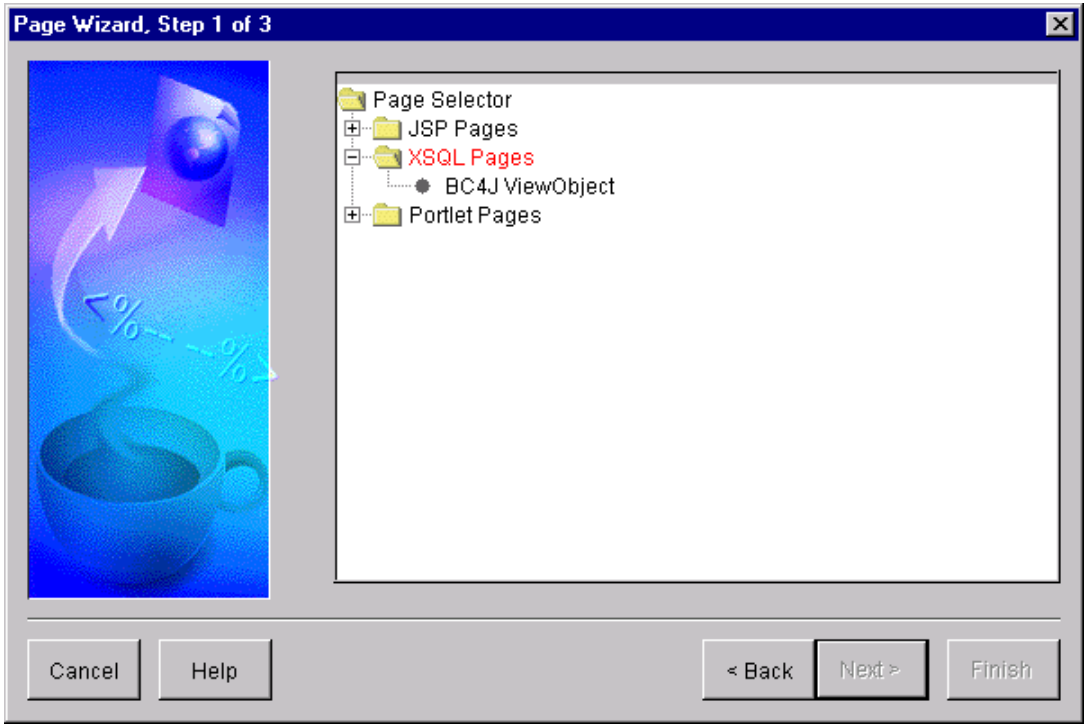


Page Selector Wizard

When a developer wants to build full-fledged XSQL pages in the process of building a web application, they can easily invoke the Page Wizard which will allow them to create XSQL Pages on top of either database tables directly or on top of BC4J View Objects. When the user chooses to build an XSQL Page on top of a BC4J View Object, they will be prompted to select an application module (Refer to JDeveloper Documentation here) from a list or create a new application module and then build the XSQL Pages based application.

Figure 14–4 illustrates JDeveloper's Page Selector Wizard.

Figure 14–4 JDeveloper's Page Selector Wizard



XML Features in JDeveloper 3.2

The following lists JDeveloper 3.2 's supported Oracle XML Developer's Kit for Java (XDK for Java) components:

- Oracle XML Parser for Java
- Oracle XSQL Servlet

You can use the XML Parser for Java including the XSLT Processor and the XML-SQL Utility in JDeveloper as all these tools are written in Java. JDeveloper provides these components.

Sample programs which demonstrate how to use these tools can be found in the [JDeveloper]/Samples/xmlsamples directory.

Oracle XDK and Transviewer Beans Integration

Oracle XDK for Java consists of the following XML tools:

- XML Parser for Java
- XML- SQL Utility for Java
- XML Java Class Generator
- XSQL Servlet
- XML Transviewer Beans

All these utilities are written in Java and hence can easily be dropped into JDeveloper and used 'out of box'. You can also update the XDK for Java components with the latest versions downloaded from Oracle Technology Network (OTN) at http://technet.oracle.com/tech/xml.

Oracle XDK for Java also includes the XML Transviewer Beans. These are a set of Java Beans that permit the easy addition of graphical or visual interfaces to XML applications. Bean encapsulation includes documentation and descriptors that make them accessible directly from JDeveloper. You can drop these beans into the TOOLS palette and use them to build applications such as XML/XSL editors.

> **See Also:** Chapter 20, "Using XML Transviewer Beans" for more information on how to use the Transviewer Beans.

Oracle XML Parser for Java

Including the Oracle XML Parser for Java in your project allows you to write applications that can search and process XML documents. You can include the

Oracle XML Parser in your project with one click as JDeveloper has a built-in library for it.

Code Insight makes understanding and using the code easier and in-place access to JavaDoc on the classes for reference. The XML parser for Java facilitates processing an XML document using either of the following interfaces:

- DOM: a tree of W3C DOM
- SAX: a stream of SAX events

Oracle XSQL Servlet

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

The XSQL Servlet offers a productive and easy way to get XML in and out of the database. Using simple scripts you can:

- Generate simple and complex XML documents
- Apply XSL Stylesheets to generate into any text format
- Parse XML documents and store the data in the database
- Create complete dynamic web applications without programming a single line of code

JDeveloper XSQL Example 1: emp.xsql

For example, consider the following XML example:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="emp.xsl"?>
<FAQ xmlns:xsql="urn:oracle-xsql" connection = "scott">
  <xsql:query doc-element="EMPLOYEES" row-element="EMP">
    select e.ename, e.sal, d.dname as department
    from dept d, emp e
    where d.deptno = e.deptno
  </xsql:query>
</FAQ>
```

Generates the following:

```
<EMPLOYEES>
  <EMP>
```

```
    <ENAME>Scott</ENAME>
    <SAL>1000</SAL>
    <DEPARTMENT>Boston</DEPARTMENT>
  </EMP>
  <EMP>
  ...
</EMPLOYEES>
```

With JDeveloper 3.2 you can easily develop and execute XSQL files. The built in Web Server and the user's default Web Browser will be used to display the resulting pages.

Using ActionHandlers in XSQL Pages

XSQL ActionHandlers are Java classes which can be invoked from XSQL Page applications very easily. Since these are Java classes they can be debugged from JDevloper just like any other Java application.

If you are building an XSQL Pages application, you can make use of the XSQL Action Handler to extend the set of actions that can be performed to handle more complex jobs. You will need to debug this ActionHandler.

Your XSQL Pages should be in the directory specified in the Project Property "HTML Paths" settings for "HTML Source Directory".

To debug your ActionHandler carry out these steps:

1. Assume you have created an .xsql file which has reference to a custom ActionHandler called MyActionHandler.
2. Debug this ActionHandler because it is not exactly behaving as you expect.
3. Set breakpoints in your Java source file.
4. He right mouse clicks on the .xsql file and now chooses Debug... from the menu.

See Also: *The JDeveloper Guide* under the online HELP menu.

XML Data Generator Web Bean

Oracle JDeveloper has an XML Data Generator Web Bean. It generates XML containing the data from a View Object and renders it to the output stream of a JSP response.

You can author JSP pages that use XML and XSL to render a response to the client.

This XML Web Bean can be used in JSP and Servlet applications. It reads data from a Business Component (View Object) and produces the appropriate XML. The strength of this Web Bean is that it analyzes the Business Component Application and navigates through it's hierarchy to produce the nested XML.

The XML Web Beans also allows the specification of an XSL Stylesheet. In addition to XML, the Web Bean can then generate HTML, WML, transformed XML and any other text format.

Mobile Application Development with Portal-To-Go and JDeveloper

Portal-To- Go and Oracle JDeveloper together offer an extremely powerful environment for developing mobile applications. Developers can use JDeveloper to generate XML from the database or from a Business Components for Java Application and use Portal-To-Go to deliver content to Web browsers, PDAs, or Cell phones.

Building XML Applications with JDeveloper

Consider the following example that demonstrates how XML is used to represent data, not present it. It shows the many to one relationship between employees and departments.

JDeveloper XML Example 1: BC4J Metadata

```
<Departments>
<Dept>
   <Deptno>10</Deptno>
   <Dname>Sales></Dname>
   <Loc>
   <Employees>
     <Employee>
        <Empno>1001></Empno>
        <Ename>Scott</Ename>
        <Salary>80000</Salary>
     </Employee>
   </Employees>
...
     </Employee>
   </Employees>
</Dept>
<Dept>
...
```

Procedure for Building Applications in JDeveloper 3.2

To build this project in JDeveloper 3.2 carry out the following steps:

1. Start a New JDeveloper Project by selecting File > New Project.
2. Create a Business Components for Java application.
3. Create an XSQL Page based upon a BC4J application module, by invoking the Page Selector Wizard. See Figure 14–4.
4. Select the application module from the list that pops up.
5. Select the View Object on which you want to base your XSQL Page.
6. Select the columns that you want to view.

When you finish these steps in the Page Wizard, you should have an XSQL Page based on the Business Components for Java (BC4J) framework View objects. When you run this page, it sends the XML data to your browser.

You could optionally create a stylesheet to format the data so that it appears in a way that you prefer or you can tune it so that it can be displayed on a PDA or cellphone.

Using JDeveloper's XML Data Generator Web Bean

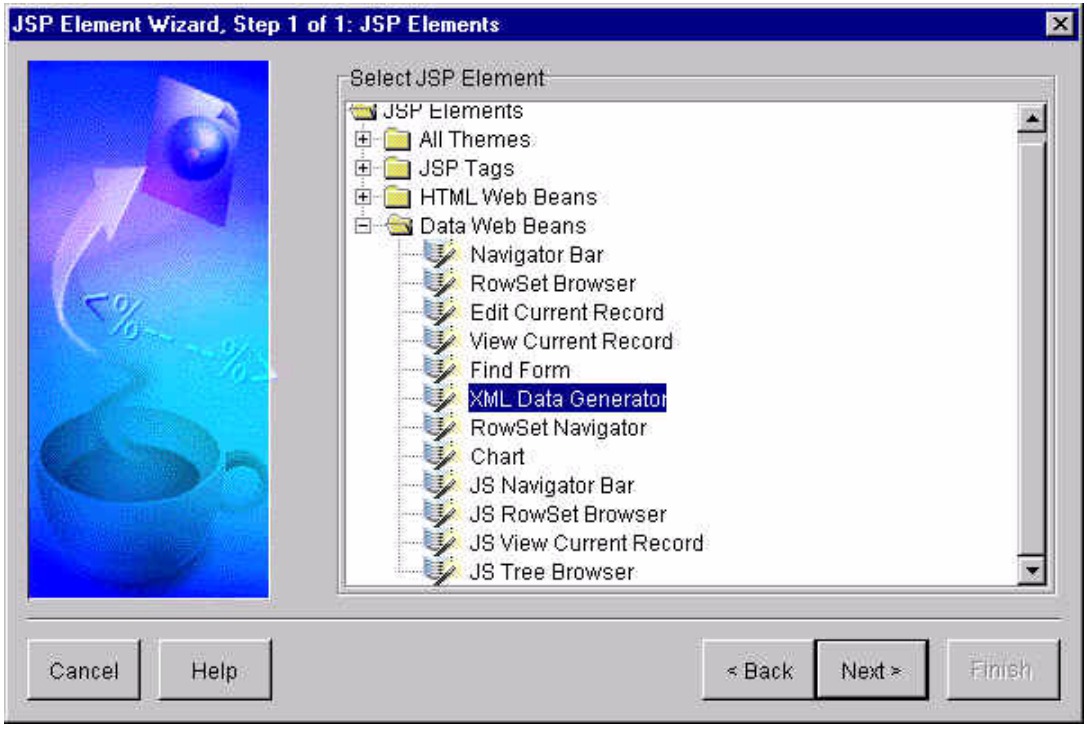
The XML Data Generator Web (Bean) can be used in JSP and Servlet applications. It reads data from a Business Component (View Object) and produces the appropriate XML. The strength of this Web Bean is the following:

- It analyzes the Business Component Application and navigates through it's hierarchy to produce the nested XML.
- It allows specification of an XSL Stylesheet. The Web Bean can then generate HTML, WML, transformed XML, and any other text format.

The Data Generator Web Bean is in the "Data Web Beans" category of the JSP Elements wizard. Figure 14–5 illustrates accessing the XML Data Generator Web (Bean) from the JSP Element Wizard.

Call the Element Wizard from your JSP or XSQL Page by right-cliking anywhere on the Page where you want to include an element. Specify the stylesheet as a parameter in the Element Wizard.

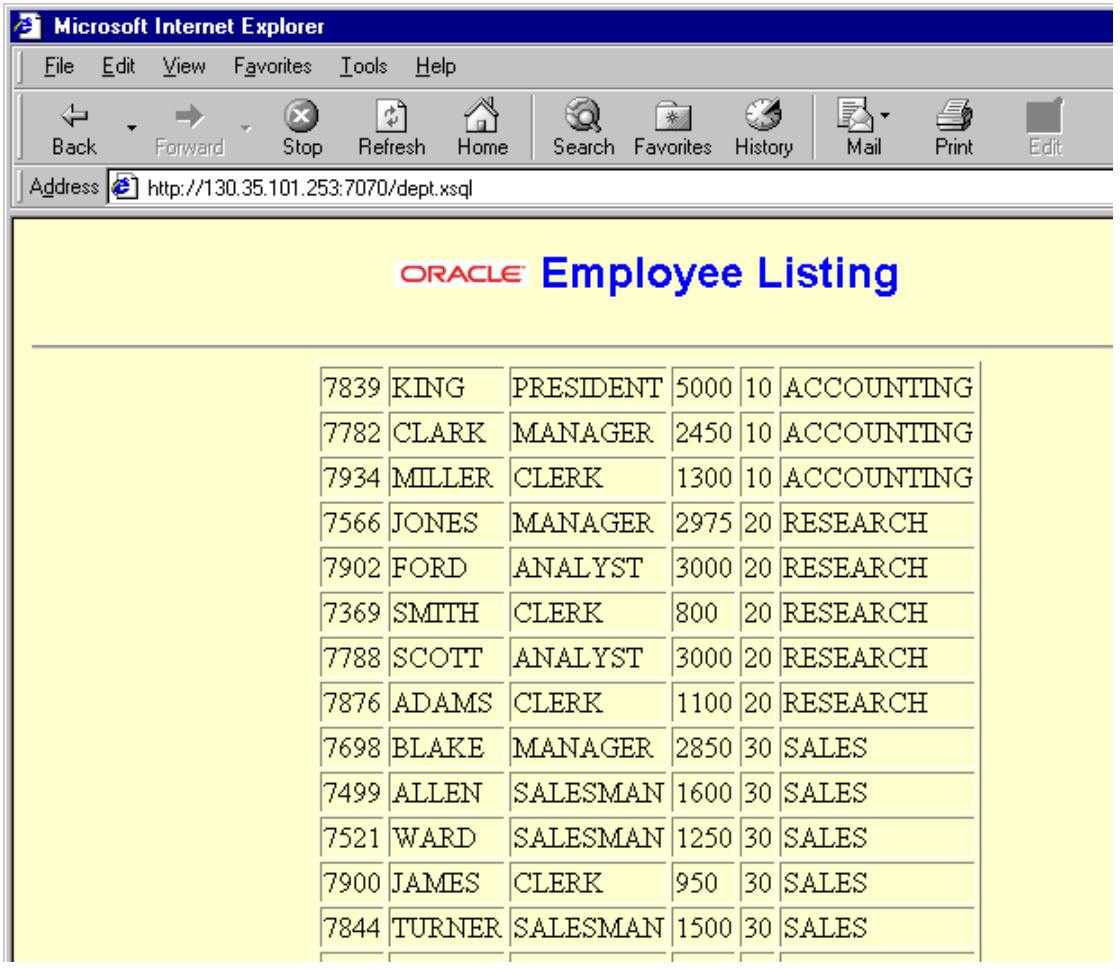
Figure 14–5 JSP Element Wizard: XML Data Generator Bean



Once you launch this wizard, you can specify a stylesheet to apply to the XML data that you generate and see the result of your output.

Figure 14–6 is an example output displayed by applying an XSL stylesheet to the employee listing.

Figure 14–6 Browser HTML Display Showing the Employee Listing (XML+XSLT= HTML)



Using XSQL Servlet from JDeveloper

XSQL Servlet offers a productive and easy way to get XML in and out of the database.

> **See Also:** Chapter 3, "Oracle XML Components and General FAQs" and Chapter 19, "Using XSQL Servlet" 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

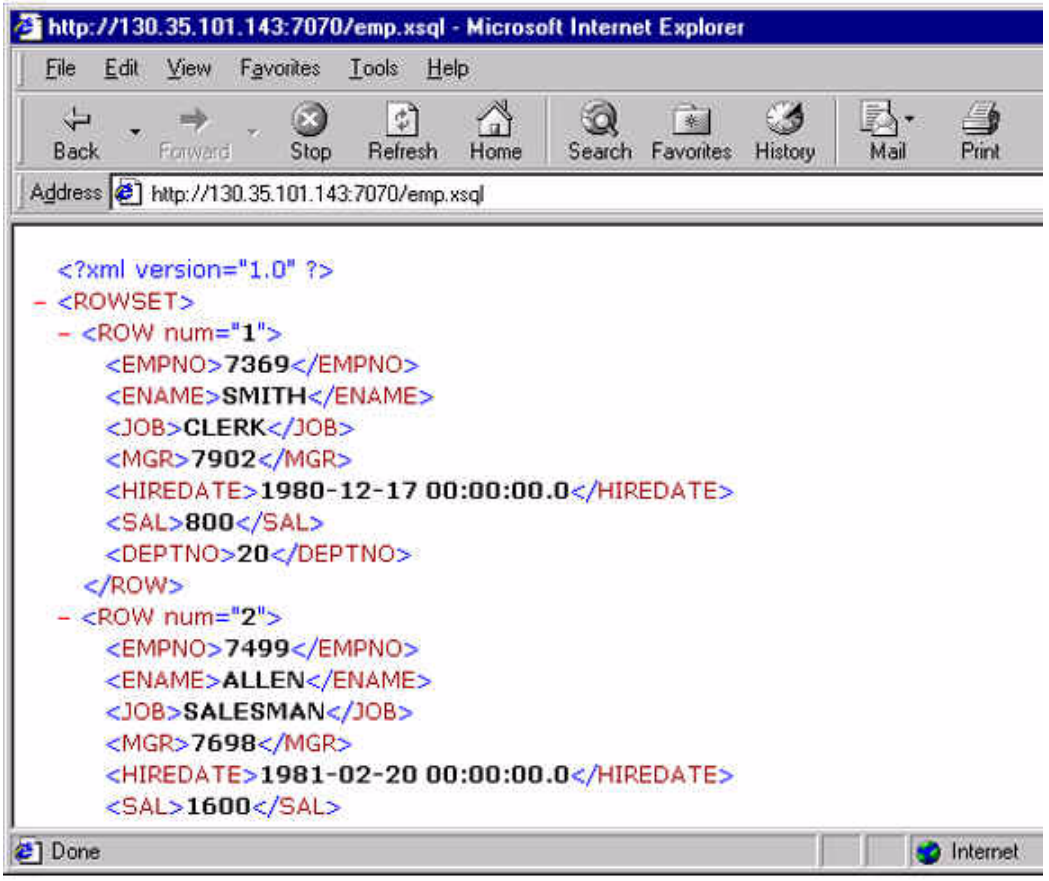
Consider a simple query in an XSQL file, which returns details about all the employees in the emp table. The XSQL code to get this information would be as shown in Example 2:

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 14–7 shows what the raw employee XML data displayed on the browser.

Figure 14–7 Employee Data in raw XML Format



If you want to output your data in a tabular form as shown in Figure 14–6, make a small modification to your XSQL code to specify a stylesheet. The changes you would make in this example are shown below highlighted.

JDeveloper XSQL Example 3: Employee Data with Stylesheet Added

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="emp.xsl"?>
<xsql:query xmlns:xsql="urn:oracle-xsql" connection="demo">
    select *
    from emp
```

```
    order by empno
</xsql:query>
```

The result would be like the table shown in Figure 14–6. You can do a lot more with XSQL Servlet of course.

> **See Also:** Chapter 19, "Using XSQL Servlet" and also the XDK for Java, XSQL Servlet Release Notes on OTN at http://technet.oracle.com/tech/xml

Creating a Mobile Application in JDeveloper

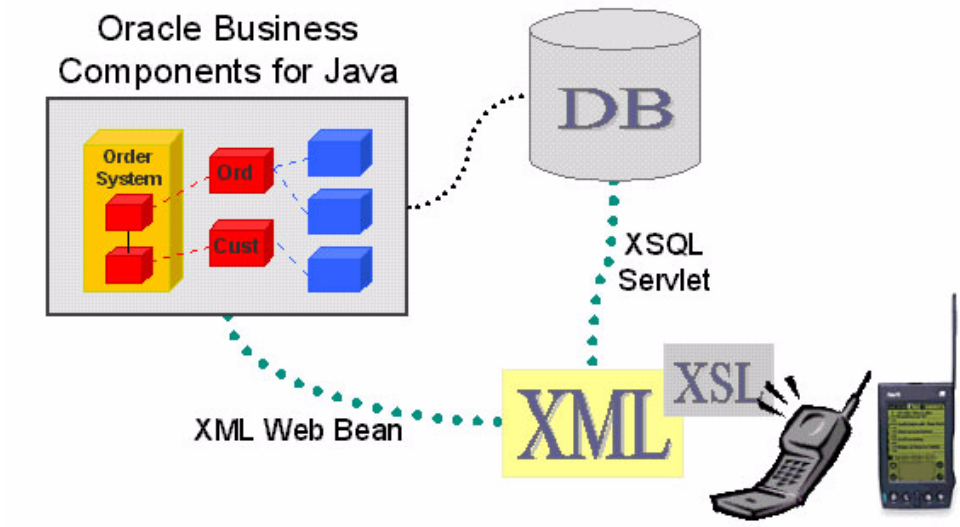
This mobile application is basically a Departments database application whose main purpose is to demonstrate how Business Components for Java (BC4J) and XML can be used to develop applications which can be accessed over wireless devices.

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, we use emulators running on Windows NT.

Figure 14–8 shows schematically how the mobile application works with BC4J, XSQL Servlet, XSL Stylesheets, and Oracle8*i*.

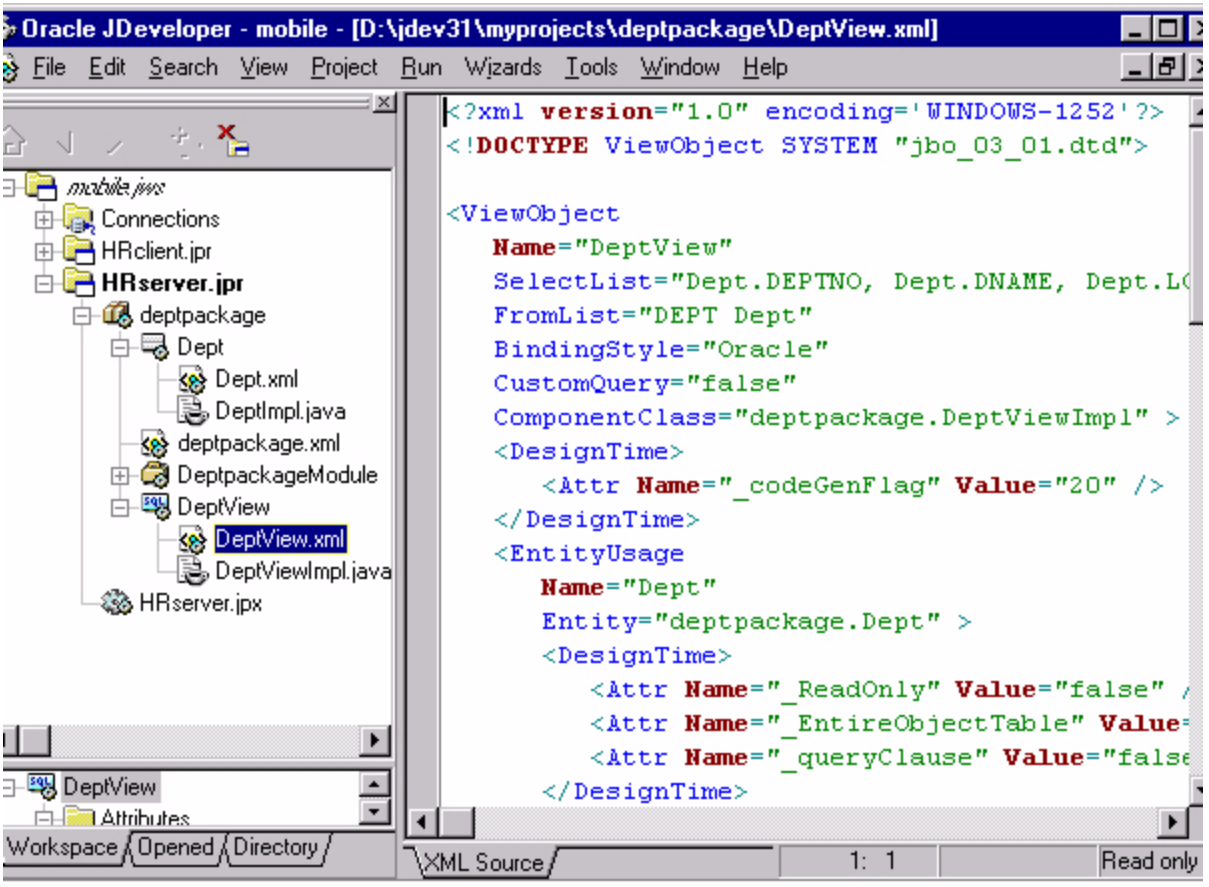
Figure 14–8 Creating a Mobile Application in JDeveloper Using BC4J and XSQL Servlet



1 Create the BC4J Application

We first create the BC4J application. It connects to the SCOTT schema on an Oracle8*i* database. Figure 14–9 shows the XML file containing the metadata about the DEPT object. See "JDeveloper XML Example 1: BC4J Metadata" on page 14-14.

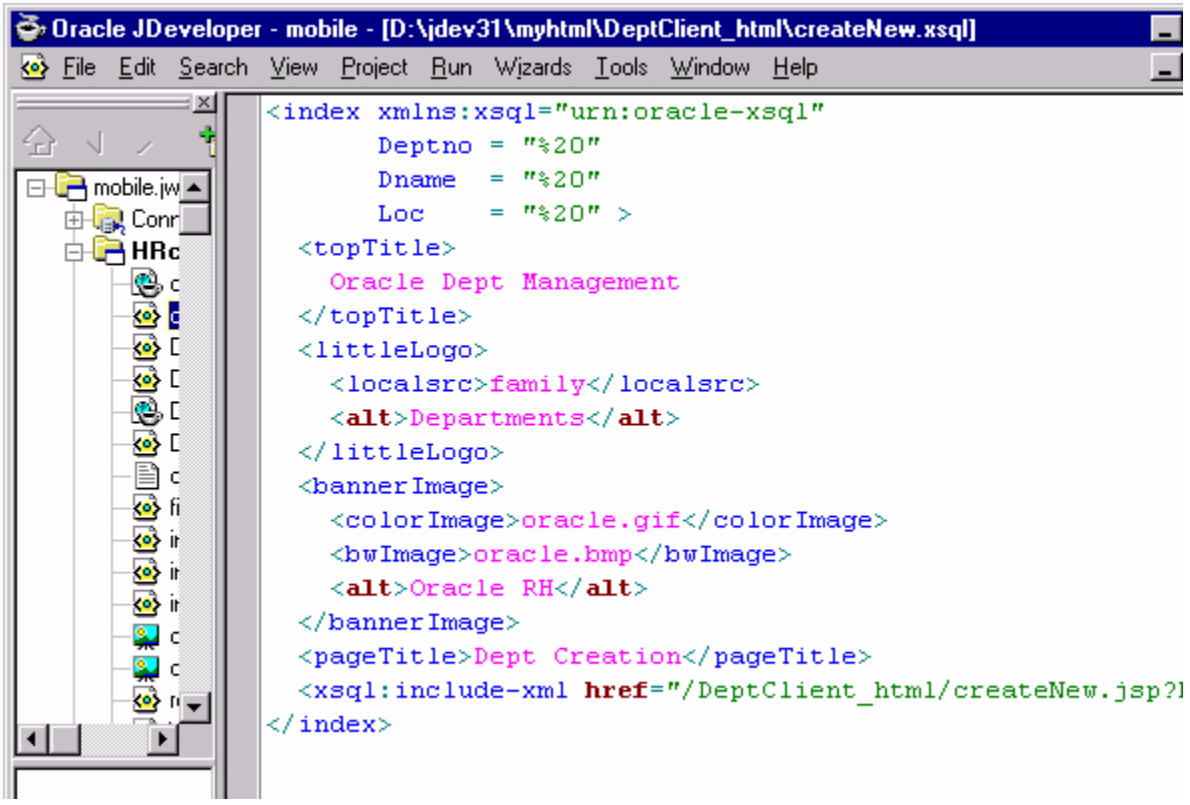
Figure 14–9 BC4J Application: DEPT View Object XML File



2 Create JSP Pages Based on BC4J Application

We then create JSP pages based upon this BC4J application. In the JSP pages we introduce the XML Data Generator Web Beans. Figure 14–10 shows the XSQL file which calls the JSP page to create the new department.

Figure 14–10 BC4J Application: XSQL File Calling JSP Page



3 Create XSLT Stylesheets According to the Devices Needed to Read The Data

We create XSLT stylesheets to go with the various client devices that we are going to access our data from. In your XSQL files, you specify the list of stylesheets and the protocols they go with which basically ties the stylesheets to the client device.

Figure 14–11 shows an example code snippet of a stylesheet (indexPP.xsl) which transforms the XML data to HTML for displaying on a browser on the Palm Pilot emulator.

Figure 14–11 BC4J Application: XSL Stylesheet (indexPP.xsl)

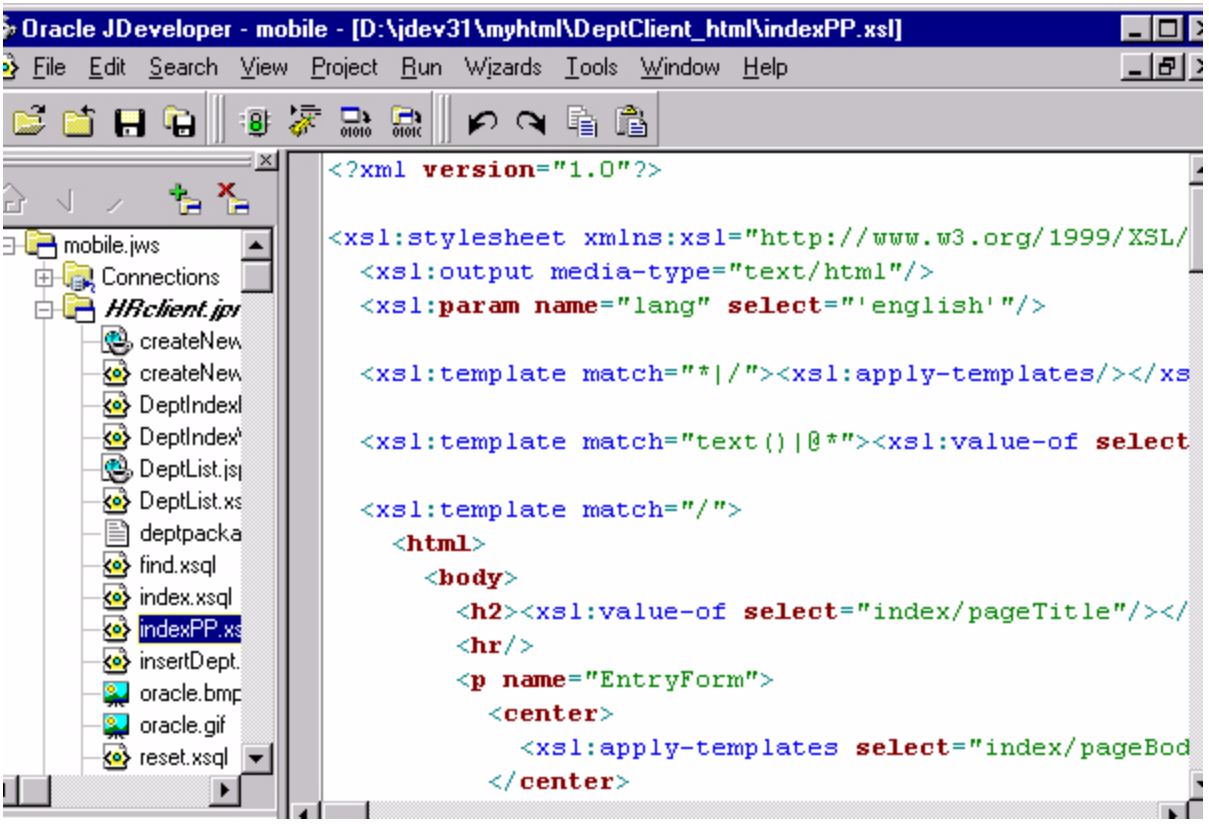


Figure 14–12 shows the Cell Phone Emulator running the Departments Application Client. It also shows the setup screen for the Cell Phone Emulator.

Figure 14–12 Cell Phone Emulator Running the Department Application Client

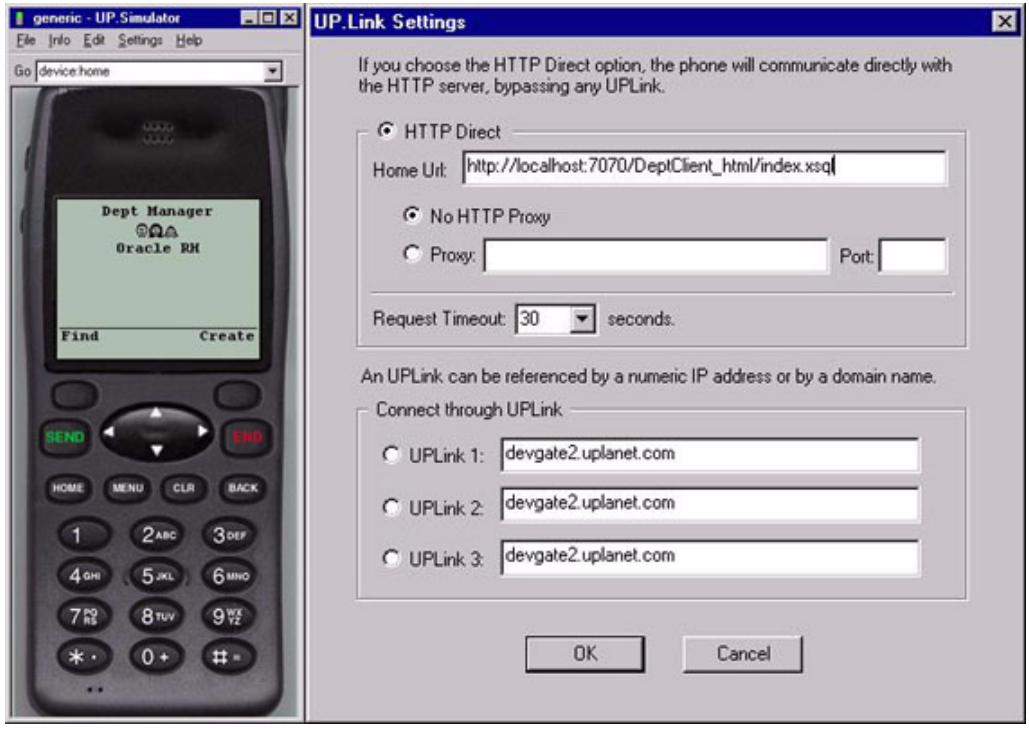


Figure 14–13 shows the Palm Pilot Emulator accessing the Departments Application via HandWeb Browser.

Figure 14–13 Palm Pilot Emulator Accessing the BC4J Departments Application Through HandWeb Browser

Frequently Asked Questions (FAQs): Using JDeveloper to Build XML Applications

Constructing an XML Document in JSP

Question

I am dynamically constructing an XML document in a JSP page (from the results of data returned by a PL/SQL API) using the classes generated by the Class generator (based on a DTD) and then applying a XSL stylesheet to render the transformation in HTML. I see that this works fine only for the first time, i.e when the JSP is first accessed (and internally compiled), and fails every time the same page is accessed thereafter.

The error I get is:

```
"oracle.xml.parser.v2.XMLDOMException: Node doesn't belong to the current
document"
```

The only way to make it work again is to compile the JSP, by just 'touching' the JSP page. Of course, this again works only once. I am using Apache JServ.

How can this be overcome? Does the 'static' code in the Java class generated for the top level node have to do anything with it?

Answer

It seems to me that you may have stored some "invalid" state in your JSP. And the XML Parser picks this "invalid" state, then, throws the exception you mentioned.

As far as I know, CRM does not use an HTTP session in their application. I guess this is tru in your case also. You may have used a member variable to store some "invalid" state unintentionally. Member variables are the variables declared by the following syntax:

```
<%! %>
```

For example:

```
<%! Document doc=null; %>
```

Many JSP users misunderstand that they need to use this syntax to declare variables. In fact, you do not need to do that. In most of cases, you do not need a

member variable. Member variables are shared by all requests and are initialized only once in the lifetime of the JSP.

Most users need stack variables or method variables. These are created and initialized per request. They are declared as a form of scriptlet as shown in the following example:

```
<% Document doc=null; %>
```

In this case, every request has its own "doc" object, and the doc object is initialized to null per request.

If you do not store an "invalid" state in session or method variables in your JSP, then there may be other reasons that cause this.

Using XMLData From BC4J

Question

I am using XmlData to retrieve data from a BC4J. I Do not use XmlData from a JSP, but from a standalone java application. In the record I target, I have the value 'R & D'.

XmlData returns 'R & D', which is fine for HTTP, but not for my needs. Can XmlData not escape the characters, and just return what's in the database ?

Answer

XmlData builds an in-memory DOM, so it must be the XML parser's serialization that's doing this. The only way I know is to do the following:

1. Write your own serializer for the DOM tree that does what you want.
2. Do an identity transform augmented with one template to write that data with disable-output-escaping="yes"

Running XML Parser for Java in JDeveloper 3.0

Question

I have downloaded JDeveloper 3.0 on my laptop (Windows 95 operating system). I am trying to run a sample XML parser program (SimpleParse.java). This program imports org.w3c.dom.Document class. I have set CLASSPATH in autoexe.bat with correct directory. The program runs on DOS prompt with "java SimpleParse

<filename>" command. I am trying to run the same program through JDeveloper but it gives me following error:

```
"identifier org.w3c.dom.Document not found"
```

Am I missing something ?

Answer

Make sure to include the Library named:

"Oracle XML Parser 2.0"

is in your project. This library is pre-defined with JDev 3.0 and you just need to visit the Project | Properties... and look at the "Paths" tab to see your project's library list.

Click the (Add...) button and pick the above library from the list.

The org.w3c.dom.* interfaces are included in this Jar. They come from the W3C and define the Document Object Model standard API's for working with a tree of XML nodes.

Question 2

Now, if I wish to use the @code as a key, I use

```
<xsl:template match="aTextNode">
    ...
   <xsl:param name="labelCode" select="@code"/>
     <xsl:value-of
 select="document('messages.xml')/messages/msg[@id=$labelCode and
 @lang=$lang]"/>
    ...
   </xsl:template>
```

that works too, but I was wondering if there isn't a way to use the '@code' directly in the 'document()' line?

Answer 2

This is what the current() function is useful for. Rather than:

```
<xsl:param name="labelCode" select="@code"/>
<xsl:value-of
 select="document('messages.xml')/messages/msg[@id=$labelCode and
 @lang=$lang]"/>
```

you can do:

```
<xsl:value-of
select="document('messages.xml')/messages/msg[@id=current()/@code
                    and @lang = $lang]"/>
```

Question 3

And finally, another question : it is - or will it be - possible to retrieve the data stored in messages.xml from the database ? How is the 'document()' instruction going to work where listener and servlet will run inside the database ?

Answer 3

Sure. By the spec, the XSL-T engine should read and cache the document referred to in the document() function. It caches the parsed document based on the string-form of the URI you pass in, so here's how you can achieve a database-based message lookup:

1. CREATE TABLE MESSAGES(lang VARCHAR2(2), code NUMBER, message VARCHAR2(200));
2. Create an xsql page like "msg.xsql" below:

```
<xsql:query lang="en" xmlns:xsql="urn:oracle-xsql" connection="demo"
            row-element="" rowset-element="">
      select message
        from messages
       where lang = '{@lang}'
         and code = {@code}
</xsql:query>
```

3. Create a stylesheet that uses msg.xsql in the document() function like:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
  <xsl:template match="/">
  <html><body>
    In English my name is
      <xsl:call-template name="msg">
         <xsl:with-param name="code">101</xsl:with-param>
      </xsl:call-template><br/>
    En espanol mi nombre es
      <xsl:call-template name="msg">
         <xsl:with-param name="code">101</xsl:with-param>
         <xsl:with-param name="lang">es</xsl:with-param>
```

```
      </xsl:call-template><br/>
    En fran&#231;ais, je m'appelle
      <xsl:call-template name="msg">
        <xsl:with-param name="code">101</xsl:with-param>
        <xsl:with-param name="lang">fr</xsl:with-param>
      </xsl:call-template><br/>
    In italiano, mi chiamo
      <xsl:call-template name="msg">
        <xsl:with-param name="code">101</xsl:with-param>
        <xsl:with-param name="lang">it</xsl:with-param>
      </xsl:call-template>
  </body></html>
  </xsl:template>
  <xsl:template name="msg">
    <xsl:param name="lang">en</xsl:param>
    <xsl:param name="code"/>
    <xsl:variable name="msgurl"
select="concat('http://xml/msg.xsql?lang=',$lang,'&amp;code=',$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 above 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.

Moving Complex XML Documents to a Database

Question

I am moving XML documents to an Oracle database. The documents are fairly complex. Can an XML document and the Oracle Development Kit (XDK) generate a possible DDL format for how the XML Document should be stored in the database, ideally generating an Object-Relational Structure. Does anyone know of a tool that can do this?

Answer a

The best way may be to use the Class Generator. Use XML-SQL Utility if DTD files are not already created. You'll still have to write a mapping program.

Another method is to create views and write stored procedures to update multiple tables. Unfortunately, you'll have to create your tables and views beforehand in either case.

BC4J Business Components for Java (BC4J) framework provides a general, meta-data-driven solution for mapping E-Commerce XML Messages into and out of the database. BC4J has a technical whitepaper on its features available at http://technet.oracle.com/products/jdev/info/techwp20/wp.html.

It is a Pure-Java, XML-Based business components framework for making building E-Commerce applications easier. It is a Java framework usable on its own, but also has tight development support built-into JDeveloper 3.0 IDE, available for download from http://technet.oracle.com/software/htdocs/devlic.htm.

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.

15

Using Internet File System (iFS) to Build XML Applications

This chapter contains the following sections:

Introduction to Internet File System (iFS)

Internet File System (*i*FS) facilitates organizing and accessing documents and data using a file- and folder-based metaphor through standard Windows and Internet protocols such as SMB, HTTP, FTP, SMTP, or IMAP4.

*i*Fs aids in the building and administering of web-based applications. It is an application interface for Java and can load a document, such as a Powerpoint .PPT file, into Oracle*8i*, and display the document from a web server, such as Oracle8*i* JVM (JServer2.0).

Working with XML in *i*FS

*i*FS is a simple way for developer's to work with XML, where *i*FS serves as the repository for XML. *i*FS can perform the following tasks on XML documents:

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

When a file is requested, it delivers select information, for example, on the web.

iFS also supports an extensible way of defining new file types and provides built-in support for defining, parsing, and rendering file types that are XML documents.

Rather than just store the XML data, Oracle *i*FS makes it possible to unlock the full potential of XML for implementing business intelligence, generating dynamic content, and sharing data across applications.

Supply a Document Descriptor

In *i*FS, when registering an XML-based file type, you supply a document descriptor that specifies the following:

- Your file type's XML document structure
- How it should be stored in the database

For example, you can save your document in its complete form in a Large Object (LOB) in the database.

Oracle *i*FS document descriptors use an XML-based syntax to describe the structure (or "schema") of its XML-based file types.

When a file is saved or sent to *i*FS, it recognizes the document as one of your file types, parses its XML, and stores the data in tables as you have specified in the document descriptor.

The same information is used to render, that is, reassemble for delivery, the XML document when a particular instance of your filetype is requested through any *i*FS supported protocols.

For more information see http://technet.oracle.com/products/ifs/

See Also: *Oracle Internet File System Developer's Guide*.

Using the *i*FS Parsers

When you drop a new XML file, whose structure *i*FS understands, into a folder, Oracle XML Parser for Java can dissect the XML file and store the separate attributes in the *i*FS schema. By defining an *i*FS subclass, you are specifying which attributes go into which columns. You can also let *i*FS make a default attribute-to-column mapping.

Once parsed, the attributes originally *within* the file become attributes *of* the file, This is extra metadata that you can edit and use as search criteria in the file system.

Consider an XML-based company standard insurance claim form. You can instruct *i*FS to parse the XML insurance claim files, extracting the attribute tag information from each file and storing these chunks separately in a table.

You can then search on the XML attributes, such as region or agent, as you would for any attribute in a file. The data is now also available for use by a relational application, such as insurance industry analytical tools.

Because the XML file has been parsed does not mean that this is the end of that file. When someone opens the original XML file, the XML renderer reconstructs it. See "Rendering XML in iFS" on page 15-7.

Standard *i*FS Parsers and Custom Parsers

In *i*FS your XML application will require a custom parser if the document format produced by the application needs it.

When Must *i*FS Parser be Explicitly Invoked?

*i*FS parser is invoked depending on how the documents produced by your application are entered into *i*FS.

- If the XML documents are uploaded using protocol servers, *i*FS XML parser will be invoked automatically by the protocol servers.

- If the XML documents are uploaded by an application, you must explicitly invoke either the *i*FS XML parser or a custom parser.If not, the XML documents will be read in as raw data, instead of being parsed into objects.
- If your application defines a custom class that produces documents in a format other than XML, create a custom parser using the classes and methods provided as part of the *i*FS Java API.

  The custom parser will create *i*FS repository objects of your custom class.

  For example, assume you have defined a Memo class that subclasses the Document class. The Memo class includes the following custom attributes: To, From, Date, and Text (the content of the memo).

  To store Memo objects in *i*FS requires a parser.

  - If the Memo documents are in XML, you can use the *i*FS `SimpleXmlParser()` to extract the attributes
  - If the Memo documents are stored in a special format, create a custom parser and specify how to extract the attributes

Using *i*FS Standard Parsers

*i*FS offers several standard parsers for creating applications in *i*FS. Table 15–1 lists the *i*FS standard parser classes.

Table 15–1 iFS Standard Parser Classes

Class	Description
SimpleXmlParser	Creates an object in the *i*FS repository from an XML document body. Used as the default parser for all XML documents stored in *i*FS. SimpleXmlParser extends XmlParser.
XmlParser	A base class for custom XML parser development.
SimpleTextParser	Provided as a starting point for developers who need to create a custom parser. The SimpleTextParser uses a simple name=value syntax.
ClassSelectionParser	Adds custom attributes to all files of a specified format. Performs no actual parsing.

Parsing Options

The following parsing options are provided by *i*FS:

- *SimpleXmlParser* for minimal customization. It works for FTP, SMB, the Windows interface, and the *i*FS Web user interface using Upload via Drag and Drop.
- *ClassSelectionParse*r. Does not perform actual parsing. It allows you to add one or more custom attributes to files with a specific file extension, such as all .doc files, before the files are stored in the repository. Maps class to a specific file format.

Using *i*FS Custom Parsers

If parsing non-XML documents, such as .doc or .xls documents, or if you have defined a custom type, such as .vcf for Vcards, you must write a custom parser to create database objects from these documents. To create a custom parser, you can either subclass an existing *i*FS parser or create a custom class from scratch, implementing the `oracle.ifs.beans.parsers.Parser` interface.

How *i*FS XML Parsing Works

When you place an XML representation of a document in *i*FS, `SimpleXmlParser()` is called to create the document object.

Assume `gking.vcf` document instance has been converted to an XML file, `gking.xml`, and that the end user is using the *i*FS Windows interface.

1. Assume next that you have dragged an instance of the XML document, such as gking.xml, into *i*FS folder, /ifs/system/vcards.
2. SMB performs a parser lookup based on the file extension, .xml. SMB is a protocol that lets you access *i*FS through Microsoft Windows 95, Windows 98, and Windows NT clients. You can drag files into and out of Oracle *i*FS, or they can edit them directly in *i*FS.
3. Because this is an XML file, the parser lookup finds a match and invokes the `SimpleXmlParser()`.
4. Because the Vcard custom class definition file was previously stored in *i*FS, the `SimpleXmlParser()` recognizes `gking.xml` as a Vcard document.
5. `SimpleXmlParser()` parses `gking.xml`, creating a Vcard object called gking.

If the gking.vcf document instance had been used instead, the result of the parser lookup in Step 2 would be that SMB would invoke the custom parser, VcardParser.

Writing a Parser Application

To write a parser application you will need to carry out the following steps

1. Write the Parser Class
2. Deploy the Parser
3. Invoke the Parser (in the parser application)
4. Write a ParserCallback (optional)

See Also: http://technet/oracle.com/products/ifs

For more information about parsers, see the following classes in the *i*FS Javadoc:

```
oracle.ifs.beans.parsers.SimpleXmlParser
oracle.ifs.beans.parsers.XmlParser
oracle.ifs.beans.parsers.SimpleTextParser
```

Rendering XML in *i*FS

By default, the *i*FS XML renderer reconstructs the original file, including the XML attribute tags. When you double-click the file, *i*FS reconstructs it, then opens it in whatever XML editor you use.

It is important to be able to reconstruct the file, but you can more creative things with the file. For example, if you parse the XML-based insurance claim, you may want to show only some sections of the original file to customers when they access it through a web-based self-service application. You may also want to calculate values, such as totals, counts, and averages, from the contents of the file, then add the results of these calculations to the file. Or you may want to change the format of the file, displaying it as RTF or HTML instead of less-readable XML.

All these tasks can be accomplished through the *i*FS renderer. The parsed contents of an XML file are waiting in *i*FS for dynamic reassembly, showing whatever range, format, or type of information you need for your users or your applications. By writing and registering a new renderer for a certain class of XML file, you can change how *i*FS displays those files.

> **See Also:** http://technet/oracle.com/products/ifs. Select "The XSL Custom Renderer Sample Application" then select "technical brief", for a detailed application how to write an *i*FS Custom Renderer.

XML and Business Intelligence

Since parsed XML files store each attribute's data in a separate, identifiable column in the *i*FS schema, Oracle *i*FS makes it possible for your data mining applications to tap into the business intelligence previously locked in your XML files.

To return to the insurance claim example, business intelligence is stored in each claim, and the aggregate value of all the information in all claims stored in the system is greater than the sum of the parts. Managers can track trends in real time as agents insert or update claim files? This is possible using *i*FS file parsing files and feeding the parsed file contents to data mining tools.

Configuring *i*FS with XML Files

To configure *i*FS, you can write XML to customize the file system. For example, a feature of Oracle *i*FS SDK is the ability to subclass files and folders. If you need to

identify your XML-based purchase orders as a separate type of file for your applications, you can define an *i*FS subclass. Creating this subclass is as simple as:

1. Writing an XML file that follows the *i*FS syntax for defining subclasses
2. Dragging and dropping this XML file into any folder in *i*FS

These types of configuration files remove the need to write complicated scripts to deploy an application. Instead, you drag and drop all the necessary files, including the XML configuration files, into the new *i*FS instance.

16

Building n-Tier Architectures for Media-Rich Management using XML: ArtesiaTech

This chapter describes the following sections:

Introduction to Building n-Tier Architectures

As broadband technologies transform the Web into a conduit for rich media, the requirements for managing multimedia content necessitate system architectures suited to address the inherent challenges therein. This chapter focuses on the tips and techniques for using XML to architect reliable, scalable and cost-effective software engineering solutions for rich-media management.

Digital resources having value are herein referred to as *digital assets*, and multimedia content, including still photographs and streaming media types such as audio and video, are designated as *rich-media*.

Digital Asset Management Challenges

The challenges of managing rich-media are immense, encapsulating a myriad of data types whose size, behavior, and access methods widely vary. Building enterprise-class applications for managing digital assets, the initial challenge is providing a massively scalable architecture capable of administering content independent of its presentation characteristics, so it may be ubiquitously expressed and exchanged across disparate systems.

Presentation Challenges

Presentation characteristic handling often necessitates specialized visualization aids (such as renderers, viewers and players) be indexed to facilitate end-user personalization. Furthermore, there are bandwidth and transport implications as rich media can be difficult to transport over local or wide area network without degrading performance.

As such challenges as emerge in a distributed, company-wide, multi-tiered, rich-media management environment, they can be directly addressed using XML.

XML-Based, Multi-Tier Communication

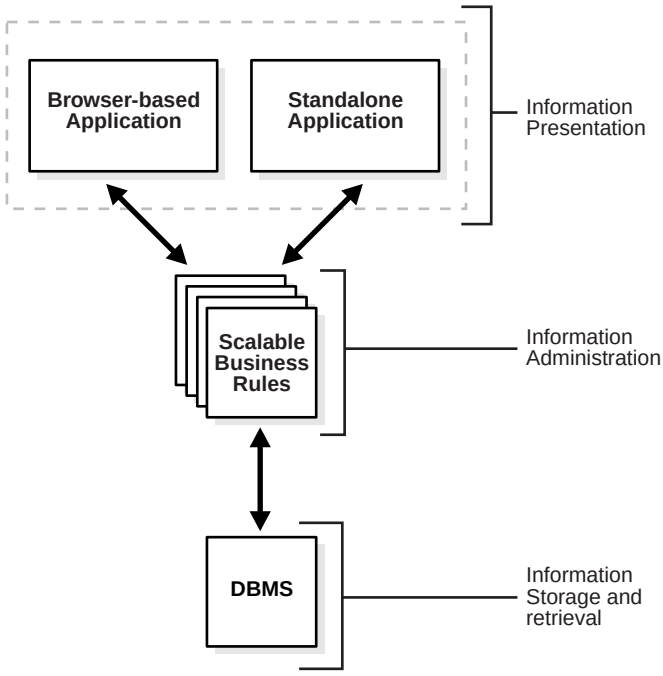
Architecting an enterprise-level digital asset management system, there are three fundamental objectives: scalability, flexibility, and extensibility. To facilitate accomplishment of these objectives, modern system architectures are generally partitioned into the following logical tasks:

- Information presentation, constituting the user interface
- Information administration, governed by business rules and procedures incorporated into application servers

- Information storage / retrieval, which is generally relegated to database management systems, such as Oracle8*i*.

Figure 16–1 shows this three-tier architecture. This is ArtesiaTech's massively scalable, enterprise-level digital asset management system called TEAMS™. It is designed and implemented on architectural underpinnings.

Figure 16–1 ArtesiaTech TEAMs 3-Tier Architecture for Digital Asset Management



Function Shipping: Separating Logical and Physical Tiers

The ability to separate the logical and physical tiers is a crucial factor to the long-term viability of a system.

In a well designed, flexible system architecture, the logical separation of the three tiers does not impose constraints on the underlying hardware. Computational logic should be able to easily be shifted onto a client machine as the necessity demands, whether it be a run-time based function for load balancing or a natural part of the

system evolution wherein functional areas are realigned to minimize the communication traffic generated across subsystems. This is known as *function shipping*.

Function shipping is the process of moving logical services to the physical areas of a system where they can be optimally run. When communication traffic between services is congested, it is often beneficial for services to reside on the same machine, although an ideal configuration is generally specific to the actual system deployment and utilization.

An essential component of function shipping is the ability to rapidly re-deploy the control logic of specific services across a myriad of disparate and frequently proprietary, operating platforms. For some, Java holds the promise to make this a reality, but ultimately, this avenue again locks the implementers into a programming language specific paradigm.

Object-oriented system behavior is ultimately governed by the messages that are passed between objects. With the goal of function shipping in mind, the method by which objects are implemented is considerably less important than how a message is serialized to be rendered cross-platform compatible and the means by which a message will ultimately be processed. XML and its derivative standards offer a suitable framework in which both of these can be effectively addressed.

Object-Oriented Messaging with XML

Messages are specifically formatted data describing events, requests and replies, and as such, messaging is the process of exchanging messages between the functional areas of a system. XML-based messaging enable applications, or specific services, to communicate across disparate programming environments, irrespective of operating system, language, or compiler, as processes in each environment need only understand the common messaging format and protocol. At the most fundamental level, this means each environment need only access to an XML parser via some API, which are now readily available for prominent programming languages such as C, C++, and Java on all of the principal operating systems: Windows, MacOS, Solaris, and Linux.

Messages are intrinsically just data structures, which can be serialized in a variety of ways. For example, consider the following C++ data structure, an array of five (5) integers:

```
int array[5] = {-9,0,-2,1,0};
```

Table 16–1 shows the logical data structure of the message.

Table 16–1 Object-Oriented Messaging with XML: Message Logical Data Structure

Array size	index 0 value	index 1 value	index 2 value	Index 3 value	index 4 value
5	-9	0	-2	1	0

However, physically, the actual bits, on an 8-bit machine are stored as shown in Table 16–2.

Table 16–2 Object-Oriented Messaging with XML: Message Physical Data Structure

Array size	index 0 bits	index 1 bits	index 2 bits	Index 3 bits	index 4 value
00000101	11110111	00000000	11111110	00000001	00000000

Now, if this array is passed as part of a message from object-1 residing on machine A to object-2 persisting on machine B, special care must be taken to assure that the destination object's view of this data is identical to the source object's interpretation, as this is by no means a certainty.

For example, the default definition of an integer on machine two may be unsigned, in which case, the logical view of the data structure is shown in Table 16–3.

Table 16–3 Object-Oriented Messaging with XML: Message Logical Data Structure With Default Integer Unsigned

Array size	index 0 value	index 1 value	index 2 value	Index 3 value	index 4 value
5	247	0	254	1	0

Furthermore, on some platforms, integers are multi-byte, 16-bit, 32-bit, and soon, if not already, 64-bit. Add to this the big endian / little endian uncertainties, and quickly, one discovers that language, compiler, or platform specific message serializations are not solutions conducive to promoting function shipping.

The XML serialization of the same array is very straightforward.

```
<array size='5'>
  <integer value='-9'/>
  <integer value= '0'/>
  <integer value='-2'/>
```

```
  <integer value= '1'/>
  <integer value= '0'/>
</array>
```

Should XML be Used for Messaging?

The natural question to ask is of course, "should XML be used for messaging?" Historically, definitions of data formats intended for cross-platform transport were specified using Interface Definition Languages (IDLs) in conjunction with middleware solutions like CORBA. In difference to XML, which incurs the nominal overhead of self-describing, human-readable data, IDLs define packets of transient, machine-readable data.

However, while XML and IDLs both provide techniques to save the state of objects, it is often desirable to have a human-readable, and thus editable, persistent state format for objects, as serialized component may be rendered inoperable by changes to class definitions.

- Using IDLs, serialized objects must be accessed via class methods
- Using XML, which is no less self-describing than as Java program or a database schema, object data can be easily examine if / when the software fails.

Using XML or IDL?

As to the question of when to use IDLs and when to use XML, opinions vary. Some argue that using XML to bind together components of a distributed system is expensive and pointless. Nevertheless, one cannot ignore that XML provides a platform neutral, programming language independent, self-describing, and thus self-documenting, mechanism for encoding object persistence.

Very few would argue that all application programs should be written in assembly language, as opposed to third or fourth generation languages, just because it is more efficient. Similarly, it should not be argued that IDLs should supplant XML in all cases as an object-oriented messaging format. Profilers can be used to judiciously apply each based upon the issues facing a software developer: time / cost ratio for implementation, maintainability, performance,...

Using XML Serialization as Part of Your User Interface Strategy

Software developers at Artesia Technologies employ XML serialization as a part of its user interface deployment strategy, as an alternative to the *object serialization* provided in the Java Swing library. Historically, subsequent versions were

incompatible with serializations produced by preceding versions. In serializing GUI components to XML, this issue has been eliminated.

Among the numerous standards that exist for XML serialization of objects are the following:

- Bean Markup Language (BML)
- XML BeanMaker
- XML-CORBA Link (XORBA)
- Koala Object Markup Language (KOML)
- Coins, an XML-based alternative to JavaBeans

These, as well as product specific formats, such as Artesia Technologies' XML Persistent Interchange Syntax for Assets (PISA), can readily be transported using any number of protocols, including RPC, RMI, and HTTP.

Message Processing with XML

To justify serializing object-oriented messages in XML, there ultimately must be recognized cost benefits to doing so. The software engineering discipline has come to recognize that actual development of code represents but a small fraction of the actual life-cycle costs. Code maintenance will inevitably accounts for the vast majority of outlays; thus, wherever possible, organizations should seek to acquire standards-based software for integration rather than writing proprietary, in-house solutions.

For example, consider a data structure for a message representing N number of digital assets, each of which contains metadata and content. This could be expressed using an ASCII-based, proprietary format as such:

```
[ASSET/METADATA]
NAME=Daytona 500
DATE=February, 2000

[ASSET/CONTENT]
FILE=daytona.jpg
```

Advantages

This approach has a number of distinct disadvantages. The processing of data formats ultimately requires two (2) types of validation:

- Structural
- Datatype

Structural validation rules specify thing covers ordering of data members, cardinality. Content validation requires fields to be validated to warrant conformance to prescribed parameters:

- Is it a date field? If so, is the date specified valid (no February 31st allowed)? Is it Y2K compliant?
- Is the field numeric? If so, does it conform within prescribed limits?
- Is the field variable character data? If so, does it have an upper bound on its length?
- Is the field an enumerated type? If so, what are the valid values?

Even more fundamental are the following issues. Where does one field end and the next begin? Where does one section [field grouping] end and the next begin. How is

containment defined? What white-space is significant, and how should the end of each line be treated? What represents a recoverable error when parsing the data? In the ASCII based scenario above, the responsibility for the specification of behavior is on the file format designer. Employing a standards-based approach such as XML, the standard itself answers these questions. With fundamental design decisions having been answered, the software developer can focus attention to implementing a solution.

Many XML parsers are available. Current APIs for processing XML, either serially or in random access mode, are, respectively, SAX (Simple API for XML) and DOM (Document Object Model). SAX is an event driven interface wherein the programmer specifies control logic for events that may occur, and as the XML parser encounters event conditions, SAX passed control to the event handler, which invoke the code prescribed for a specific event.

Specifications for ensuring structural validity for XML data are DTDs, utility descending from SGML heritage and a sundry of schema proposals that employ the actual XML syntax to describe structural validation criteria. DTD enabled validation is available by all validating XML parsers. Schema based validation can be employed via source libraries supporting the specifics of particular schema specification.

Some schemas, such as the working draft of *XML Schema*, made available by the W3C, provide both structural and datatype validation. Ultimately, all of the strategies provide a well-defined procedure to define validation rules, which can be enumerated within or separate from the actual runtime code. Because these are self describing, the can intuitively be modified by software developers or integrators. Because these are well-defined approaches, tools availability is generally forthcoming, which translates into lines of code not written, not maintained, and subsequently, cost savings within the software development lifecycle.

Scripting With XML

Returning to the initial goal of function shipping, what is ultimately necessary is a ubiquitous practice to *describe* and *process* data. The actual logic, encapsulated within components or objects, must be shippable; however, the runtime code itself need not be when the logic controllers are data driven. XML provides a ubiquitous way to process data that can be employed to govern system behavior.

Consider the fundamentals of digital asset management. These naturally involve a specialized set of CRUD (Create, Read, Update, Delete) operations that can be performed on assets, each which constitutes a specific behavior.

Table 16–4 shows a matrix that categorizes these CRUD operations.

Table 16–4 XMl Scripting; CRUD (Create, Read, Update, Delete) Operations

create			read			update		delete
import	Version	Checkin	Search	export	Checkout	replace	delete	purge

From an object-oriented perspective, many operations are specialization of others. For example, *version* is a specialization of *import* [create], wherein a new digital asset is created and the metadata and / or content of a previous rendition is carried forward. The *check-in* operation is a specialization of *version*, with the added step to release the lock--the digital asset is assumed to be checked-out, and thus, locked prior to check-in.

XML Persistent Interchange for Assets (PISA)

Artesia Technologies' XML Persistent Interchange Syntax for Assets (PISA) defines a data structure that is composed of N digital assets, where in actual practice, the upper bound of N can sometimes literally be in the hundreds of thousands. With an XML data stream that can measure gigabytes in size, the DOM cannot be sensibly employed--the memory required to so would be immense; hence, Artesia Technologies' DAM system, TEAMS, utilizes SAX instead, with the transaction level being per asset.

To aggregate the components of a digital asset expressed within a PISA file using SAX, the default document handler must be subclassed, coded specific to the PISA format. Because the CRUD operations enumerated above and the transaction level are per asset, a generic *AssetProcessor* class can be defined from which all operations on asset are derived; hence, all operations are by definition *AssetProcessor*(s). When an [XML] end tag for an asset is encountered, the document handler will have completed the construction of the in memory data structure representing that asset, and can subsequently initiate a call to the process method of an *AssetProcessor* for that asset.

Which *AssetProcessor* is applied varies based upon the actual type passed to the constructor of the document handler, usually governed by a specific callback intended to invoke an asset-level operation; thus, writing a single XML document handler, all asset operations can be made available as bulk operations in the digital asset management system. This approach is referred to as a *design pattern*. As it simplifies the design, implementation and maintenance of the XML processing,

there being need for only a single data format [PISA] and document handler, enormous cost savings are realized within the software lifecycle.

Selection of *AssetProcessor* via a specific callback has a singular drawback, that being the processor type is fixed for the duration of the process. Often, however, it is desirable to queue processing on specific assets, each of which may have a distinct per asset operation. To process this queue as a single batch job, it is necessary for the XML file to be coded so that an event can be fired indicating a change of *AssetProcessor*. To realize this, Artesia Technologies designed an embedded scripting language that employs XML processing instructions to syntactically enumerate the control of system behaviors, such as which *AssetProcessor* is currently active.

Syntactic constraints on the internal data of a processing instruction, following its target, are minimal; therefore, data can be encapsulated in element attribute-like syntax as follows:

```
<?PITarget attr1='some value' attr2='another value'?>
```

Using SAX

As the SAX event for handling processing instructions conveniently dissects a processing instruction into its target [name] and its remaining data, parsing an element-attribute like syntax requires writing a minimalist document handler, as the processing instruction can thus be transliterated into an element and submitted to XML parsers, as follows:

```
<PITarget attr1='some value' attr2='another value'/>
```

The SAX event for handling start elements returns a list of attribute value indexable by attribute name; therefore, the processing instructions can be employed a scripting language that can easily be processed in a consistent manner, employing the XML parser as an application framework. Thus, by using XML, one can readily build a data driven, event based system capable of providing all services traditionally invoked by programmer level APIs.

Inter-Tier Communication

Digital asset management systems ultimately provide application services at the middle architectural tier that must be able to push and pull data from the persistent storage layer, which is generally proprietary, specific to the database management and/operating system. Utilizing XML an application framework, it is possible to construct a DAM system that can rapidly be redeployed as new platforms arise. This can be accomplished because XML inherits from its SGML lineage the notion of separating logical and physical data representations.

Logically, a digital asset is comprised of its metadata and its content; however, physically, these two components may be stored in any number of ways. This is where the XML conception of *entities* is useful. The concept of digital assets can be codified as follows:

```
<!DOCTYPE teams:digital-asset [
<!ENTITY metadata PUBLIC "-//TEAMS//TEXT digital asset metadata//EN"
"metadata.ent">
<!ENTITY content  PUBLIC "-//TEAMS//TEXT digital asset content//EN"
"content.ent" >
  ]>
  <teams:digital-asset>
    &metadata;
    &content;
  </teams:digital-asset>
```

As the XML parser encounters the entity references, it looks to the entity declaration to determine from where it should pull the actual entity data. This resolution of an entity reference in its declaration is handled by an entity manager. Fortunately, XML also falls heir to the notion that entity management should be kept independent from the actual parser; therefore, by substituting different entity management, numerous underlying physical storage schemes can be supported.

The most basic persistent storage mechanism for the metadata of a digital asset would be XML serialization onto a file system; hence, the entity manager would need only map the PUBLIC identifier for the metadata to a specific host, path and filename.

The same data can also be stored within relational or object-oriented databases, SGML/XML component managers, and proprietary data format schemes. Because one specific implementation is not optimal, in terms of price, performance or other variables, for all situations, the DAM system must be easily configurable to facilitate any of these persistent storage systems.

As an entity manager can be written for to retrieve data from any conceivable storage strategy, XML can be used to serve as a level of indirection between persistent storage and the business logic.

As entities are resolved, pulling data from persistent storage, the information can, again, be serialized as XML and passed to middle-tier, application services for processing. This provides a ubiquitous mechanism for processing the data independent of the plethora of known and unknown methods by which the data can be warehoused.

Transaction State Management for Web-Based Services

When providing web-based access to digital asset management services, the challenge arises to maintain and aggregate user input from numerous wizards, all of which may be optional, into a data format capable of being reliably submitted for processing. Because entities can resolve to actual or empty data streams, XML can be deployed as a skeletal structure wherein any, all, or none of the wizard delivered data can be maintained as separate physical storage units and subsequently aggregated by the XML parser.

For example, consider the scenario wherein there are three wizards, each represented by an HTML page. User input is sequentially supplied as HTML form data and submitted to a servlet for processing as the user navigates from one HTML page (wizard) to the next. In order to preserve the contents of one page to the next, there are a number of approaches that can work. The data can be persistently stored inside of cookies, or it can be passed between the pages as hidden form data, but XML offers by far a more elegant solution. Consider the following XML-based skeletal structure for a transaction:

```
<transaction>
  &wizard1;
  &wizard2;
  &wizard3;
</transaction>
```

The data for each wizard is represented by an entity reference, which resolves from some storage system, ordinarily as simple a single file. Irrespective of the sequence of wizard traversal, the departure from one page always produces a transaction codified as an XML data stream whose readiness for submission can easily be governed by DTD-assigned rules. As data does not need to be passed between pages as part of an HTTP session, the state of a transaction can be efficiently maintained and updated.

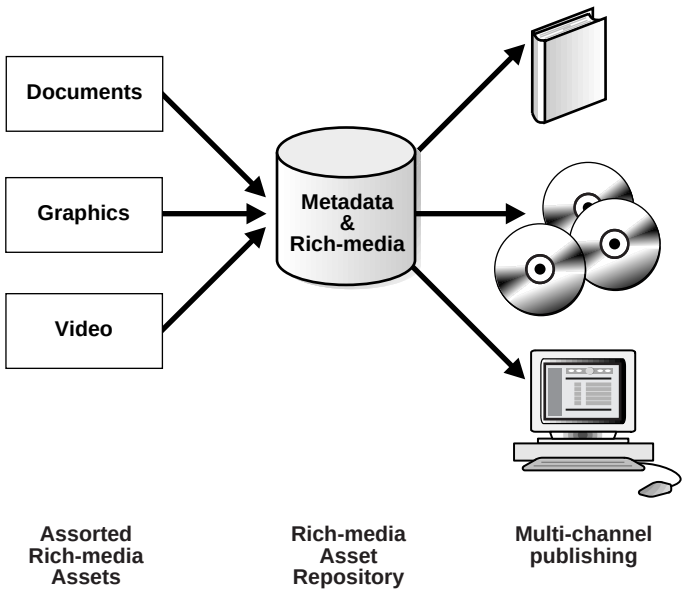
XML-Based Software Quality Assurance

One measure of success for a software developer is how a product behaves in accordance with prescribed and unexpected inputs. Determining the level of quality or rate of defects has, historically, been an extremely laborious, human intensive process. Ultimately, process flows within software represented as finite state machines, whereby given an initial state and possible set of inputs, a set of new states resulting from the input are realized. If state-transitions can be programmatically captured at runtime using XML, formal grammars such as DTD can document and assure that a program meets prescribed specifications.

XML-Based Data Dissemination

The same XML standard that serves beneficially to construct a high quality, completely scalable, enterprise digital asset management system is also equally applicable for pushing rich media content, and its associated metadata, downstream to a variety of distribution channels. As digital assets are rendered from a repository or a directory, XML serves, again, as the data glue binding metadata and content. Thereafter, one or more transformation programs can be applied to enhance / filter metadata and convert content types from one format to another. Subsequently, the digital assets can be aggregated to populate an asunder of publication vehicles, such as page layout programs, CD mastering tools or web site management software as illustrated in Figure 16–2.

Figure 16–2 XML-Based Data Dissemination



XML-Centric Digital Asset Management

The task of aggregating non-streaming, self-contained content types, such as static text or images, has been straightforwardly understood for many years. Page layout programs and modern word processors capable of rendering compound documents are abundant, but streaming media types employed in the broadcasting of audio and video have an additional set of challenges. Integrating sets of independent multimedia objects into a synchronized rich media presentation requires the ability to specify temporal (i.e. time based) behavior and layout of the presentation as well as the capacity to connect aperients within media objects.

Digital Asset Aggregation Facilitating End-User Personalization

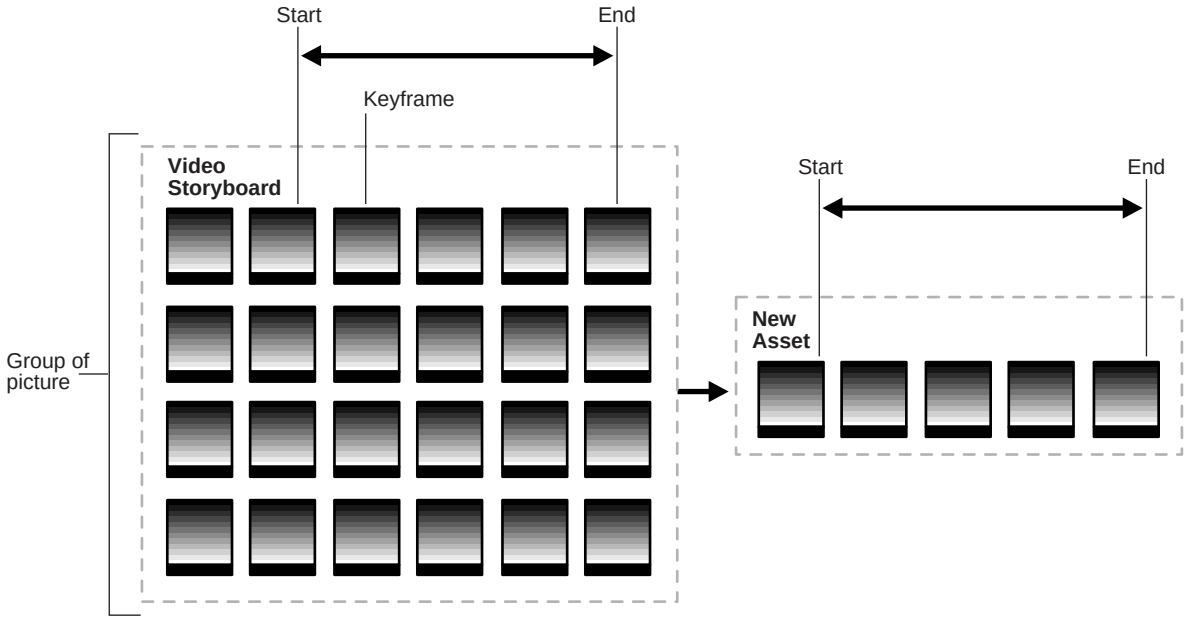
SMIL, the (Synchronized Multimedia Integration Language), is an application of XML designed to meet the specific challenges of binding multimedia objects (audio, video, still images) as a unified and synchronized rich media presentation.

Employing SMIL, the timing of media objects can be coordinated for rendering, and as each content object is identified by a URL, the rich media components of a presentation can be streamed from numerous sources.

Figure 16–3 shows how SMIL can be used to implement a play list for storyboarding video clips permitting users to select numerous video segments, adjust the in and out points thereof, and then rendering them in a seamless presentation.

This is accomplished by having the application generate a SMIL file that realizes the sequential playback of selected video clips. The user can thereafter easily refine the presentation until satisfied with the selection, cuts, and sequence of the video, facilitating generation of rough-cut edits from digital videos prior to employing an editing suite to fine-tune frame transitions.

Figure 16–3 How Users Can Select Video Clips From Video StoryBoard Using SMIL



User preferences with respect to visualization aids (such as renderers, viewers and players) employed must also be managed. XML is exceptionally well-suited to enable personalization.

As the volume of information for preferences as such is relatively insignificant, the DOM API can be generally be deployed to efficiently create and query end-user personalization.

As the preferences are encoded in XML, they can be seamless managed and interchanged through out the system irrespective of the underlying persistent storage tier. In so much as XML-based rich-media aggregation standards such as SMIL are employed, user preferences specific as such can easily be embedded with the personalization serialization with very minimal computational overhead.

Digital Asset Aggregation Addressing Bandwidth Implications

In the quest for ubiquitous mobility there are a number of standards. Two of these standards of relevance to the Digital Asset Management are described here:

- WML, the Wireless Markup Language, is an application of XML that promotes text from Web pages being rendered on mobile phones, pagers, personal digital assistants, and other wireless, mobile access devices. WML is a subset of the Wireless Application Protocol (WAP), an initiative to define an industry standard for developing applications over wireless communication networks. WML is aimed to confront constraints faced in wireless devices such as limited display and user input facilities, limited computational resources, and a narrowband network connection. Documents stored within a rich-media repository can be converted into WML by invocation of an appropriate transformer when digital assets are published from the system.
- SyncML is yet another XML application aimed at providing ubiquitous data format to synchronize devices and application of disparate networks. Using SyncML, personal data such as email, calendars, and contact information, can be synchronized to warrant consistency and accessibility irrespective of where information is stored. For example, SyncML could be used to transfer email stored within a personal digital assistant to a PC-based mail client. SyncML also seeks to minimize bandwidth, and it is tailored to address issues such as low reliability connections and high latency networks.

Summary

XML remains one of the most recommended technologies for standards-based commercial software development. Not only does XML provide an overabundance of "step-saving" features for designing real-world rich-media management software applications, it offers best-in-class interoperability amongst all the data interchange standards available.

Given the advent of rich-media in the rapidly converging industries of broadcasting, entertainment and computers, XML has uniquely positioned itself as the "data glue" in the midst of a maze of incompatible media / data formats.

Higher is the degree of incompatibility and diversity of digital mediums in near future, greater will be the value realized from the return on investment (ROI) of having implemented systems using XML as the data interchange protocol.

Coupled with the endorsements of wired as well as the wireless world, and the proven track record of aiding flexible, modular system development lifecycles, XML remains one of the favorite open technologies for the brave new world of rich media.

Part VI

XDK for Java

Part VI describes how to access and use XML Development Kit (XDK) for Java.

Part VI contains the following chapters:

- Chapter 17, "Using XML Parser for Java"
- Chapter 18, "Using XML Java Class Generator"
- Chapter 19, "Using XSQL Servlet"
- Chapter 20, "Using XML Transviewer Beans"

FAQs are included at the end of the following chapters:

- Using XML Parser for Java chapter
- Using XSQL Servlet chapter

17

Using 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 stand-alone XML component that parses an XML document (or a standalone DTD) so that it can be processed by an application. Library and command-line versions are provided supporting the following standards and features:

- DOM (Document Object Model) support is provided compliant with the W3C DOM 1.0 Recommendation. These APIs permit applications to access and manipulate an XML document as a tree structure in memory. This interface is used by such applications as editors.
- SAX (Simple API for XML) support is also provided compliant with the SAX 1.0 specification. These APIs permit an application to process XML documents using an event-driven model.
- Support is also included for XML Namespaces 1.0 thereby avoiding name collisions, increasing reusability and easing application integration.
- Able to run on Oracle*8i*, *i*AS (Internet Application Server), and OIS (Oracle Integration Server)

Additional features include:

- Validating and non-validating modes
- Built-in error recovery until fatal error
- DOM extension APIs for document creation

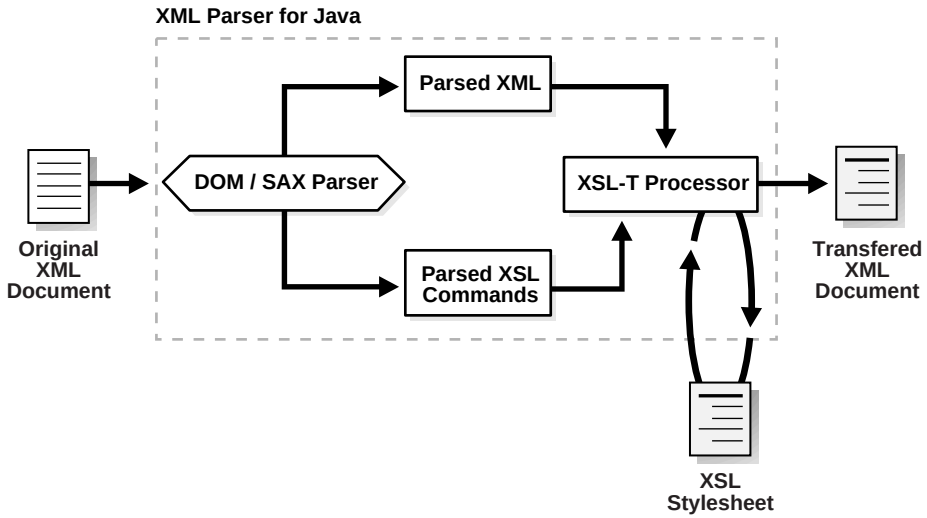
The parsers are available on all Oracle platforms.

See Figure 17–1. The XML Parser for Java inputs an XML document. 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 XSL-T Processor where the selected stylesheet is applied and the transformed (new) XML document is then output.

See Also: Appendix C, "XDK for Java: Specifications and Cheat Sheets".

Figure 17–1 Oracle XML Parser



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

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

- Figure 17–4, "XML Parser for Java: DOMParser()"
- Figure 17–5, "Using SAXParser() Class"

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

- Figure 17–6, "XSLProcessor Class Process"

XSL Transformation (XSL-T) Processor

The V2 versions of the 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, XML to HTML, or to virtually any other text-based format. See Figure 17–1.

The processor supports the following standards and features:

- Compliant with the W3C XSL Transform Proposed Recommendation 1.0
- Compliant with the W3C XPath Proposed Recommendation 1.0
- Integrated into the XML Parser for improved performance and scalability
- Available with library and command-line interfaces for Java, C, C++, and PL/SQL

Namespace Support

The Java, C, and C++ XML parsers also support XML Namespaces. Namespaces are a mechanism to resolve or avoid name collisions between element types (tags) or attributes in XML documents.

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

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

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

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

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

Validating and Non-Validating Mode Support

The Java, C, and C++ parsers can parse XML in validating or non-validating modes.

- **Non-Validating Mode**: The parser verifies that the XML is well-formed and parses the data into a tree of objects that can be manipulated by the DOM API.
- **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.

See Also: *Oracle8i XML Reference*

Parsers Access XML Document's Content and Structure

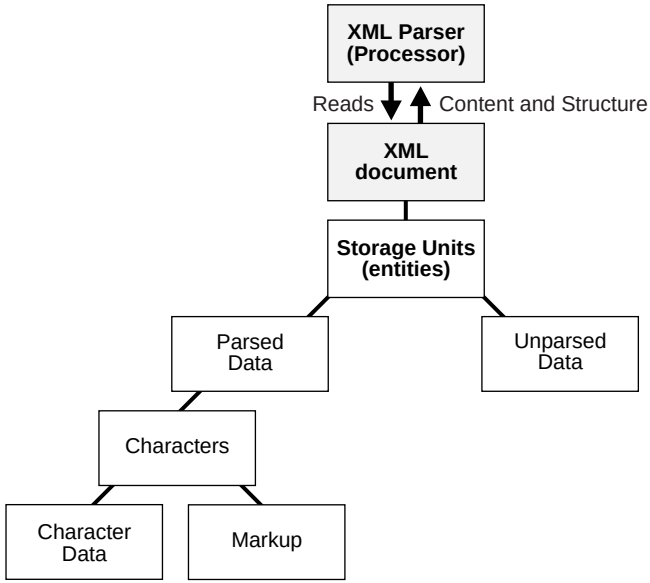
XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup.

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

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

This parsing process is illustrated in Figure 17–2.

Figure 17–2 XML Parsing Process



DOM and SAX APIs

XML APIs generally fall into the following two categories:

- Event-based
- Tree-based

See Figure 17–3. Consider the following simple XML document:

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

DOM: Tree-Based API

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

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

SAX: Event -Based API

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.

The above XML document becomes a series of linear events as shown in Figure 17–3.

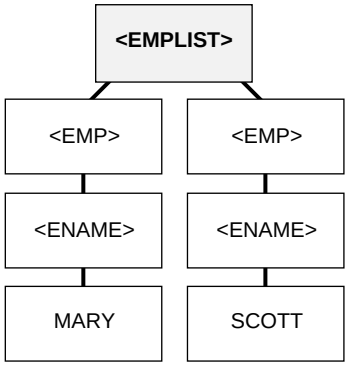
Figure 17–3 Comparing DOM (Tree-Based) and SAX (Event-Based) APIs

[

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 'rules of thumb' for using DOM and SAX APIs:

DOM:

- Use the DOM interface when 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.
- When using the DOM interface, see if you can use more attributes over elements in your XML (to reduce the pipe size).

SAX:

Use the SAX interface when data comes in a streaming manner (using some I/Pstream).

Running the XML Parser for Java Samples

Table 17–1 lists the XML Parser for Java examples provided with the XDK for Java software. The samples are located in the /sample subdirectory. They illustrate how to use Oracle XML Parser for Java.

Table 17–1 XML Parser for Java Samples

Name of Sample File	Description
DOMSample	A sample application using DOM APIs.
SAXSample.java	A sample application using SAX APIs.
XSLSample	A sample application using XSL APIs.
DOMNamespace	A sample application using Namespace extensions to DOM APIs.
SAXNamespace	A sample application using Namespace extensions to SAX APIs.

In addition, because some of the package names are different in V2, difference files were generated to show the differences between V2 and V1 of the XML Parser for Java.

To run the sample programs:

1. Use "`make`" to generate .class files.
2. Add `xmlparserv2.jar` and the current directory to the CLASSPATH.
3. Run the sample program for DOM/SAX APIs as follows:

   ```
   java <classname> <sample xml file>
   ```

4. Run the sample program for XSL APIs as follows:

   ```
   java XSLSample <sample xsl file> <sample xml file>
   ```

A few XML files such as `class.xml`, `empl.xml`, and `family.xml`, are provided as test cases.

XSL stylesheet `iden.xsl`, can be used to achieve an identity transformation of the supplied XML files:

- class.xml
- NSExample.xml

- family.xml
- empl.xml

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

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

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

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

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

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

DTD: family.dtd

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

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

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

</xsl:stylesheet>
```

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

```
<!DOCTYPE doc [
<!ELEMENT doc (child*)>
<!ATTLIST doc xmlns:nsprefix CDATA #IMPLIED>
<!ATTLIST doc xmlns CDATA #IMPLIED>
<!ATTLIST doc nsprefix: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 implement the following classes:

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

Figure 17–4 shows the main steps you need when coding with the DOMParser() class:

- *Without DTD Input*
  1. A new DOMParser() class is called. The available properties to apply to this class are:
     - setValidateMode
     - setPreserveWhiteSpace
     - setDocType
     - setBaseURL
     - showWarnings
  2. This is applied to XMLParser.parse() along with the XML input. The XML input can be a file, string buffer, or URL.
  3. The XMLParser.getDocument() method is implemented.
  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

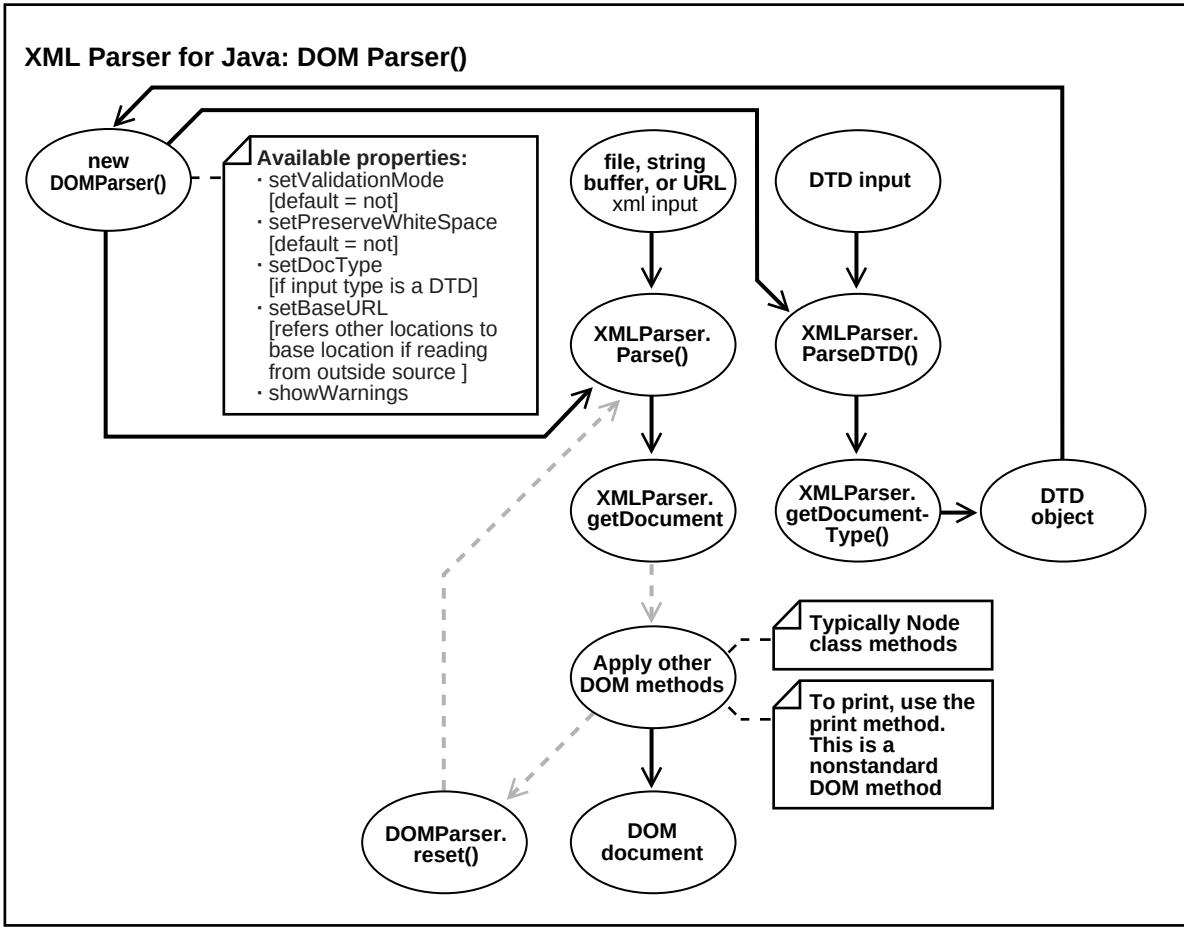
- setDocType
- setBaseURL
- showWarnings

2. This is applied to XMLParser.parseDTD() method along with the DTD input.
3. The XMLParser.getDocumentType() method then sends the resulting DTD object back to the new DOMParser() and the process continues until the DTD has been applied.

Using the DOMParser() class is illustrated with the following example:

- "XML Parser for Java Example 1: Using the Parser and DOM API (DomSample.java)"

Figure 17–4 XML Parser for Java: DOMParser()



XML Parser for Java Example 1: Using the Parser and DOM API (DomSample.java)

```
// 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(true);
         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 17–4. 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(true);
   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 aURL. 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 17–3 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;
    }
}
```

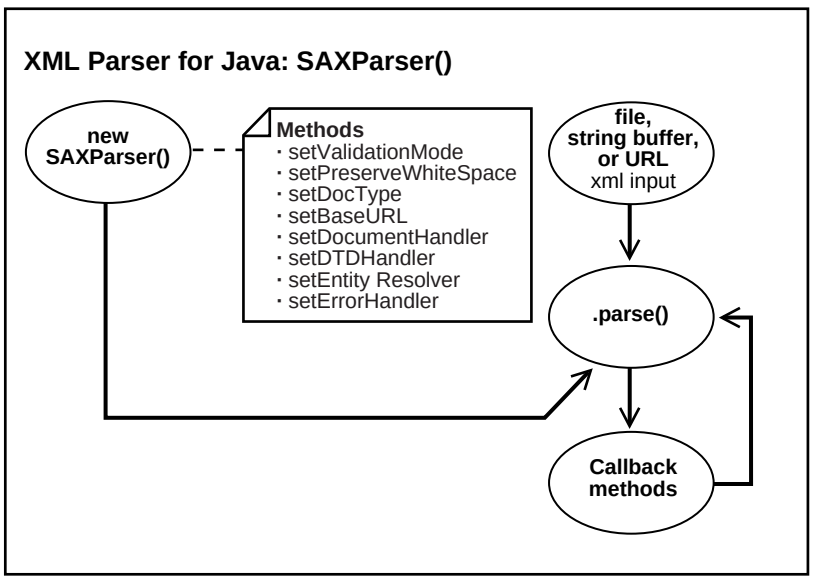
Note: No DTD is input is shown in Example 2.

Using XML Parser for Java: SAXParser() Class

Figure 17–5 shows the main steps you need when coding with the SAXParser() class. The SAXParser() class is implemented in the following example:

- "XML Parser for Java Example 3: Using the Parser and SAX API (SAXSample.java)"

Figure 17–5 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());
    }
}
```

Using XML Parser for Java: XSL-T Processor

To implement the XSL-T Processor in the XML Parser for Java use XSLProcessor class.

Figure 17–6 shows the overall process used by the XSLProcessor class.

1. A new XSLProcessor() class declaration begins the XSL-T process.
2. There are two inputs:
   - "**Stylesheet**". First a stylesheet is built. A `new XSLStylesheet()class` is declared with any of the following available methods:
     * removeParam()
     * resetParam()
     * setParam()
   - "**XML input**". This can can repeat 1 through n times for a particular stylesheet. This inputs the "Process Stylesheet" step.

   Both inputs can be one of four types:
   - input stream
   - URL
   - XML document
   - Reader
3. The resulting stylesheet object and the XML input, feed the "Process Styesheet" step, namely:

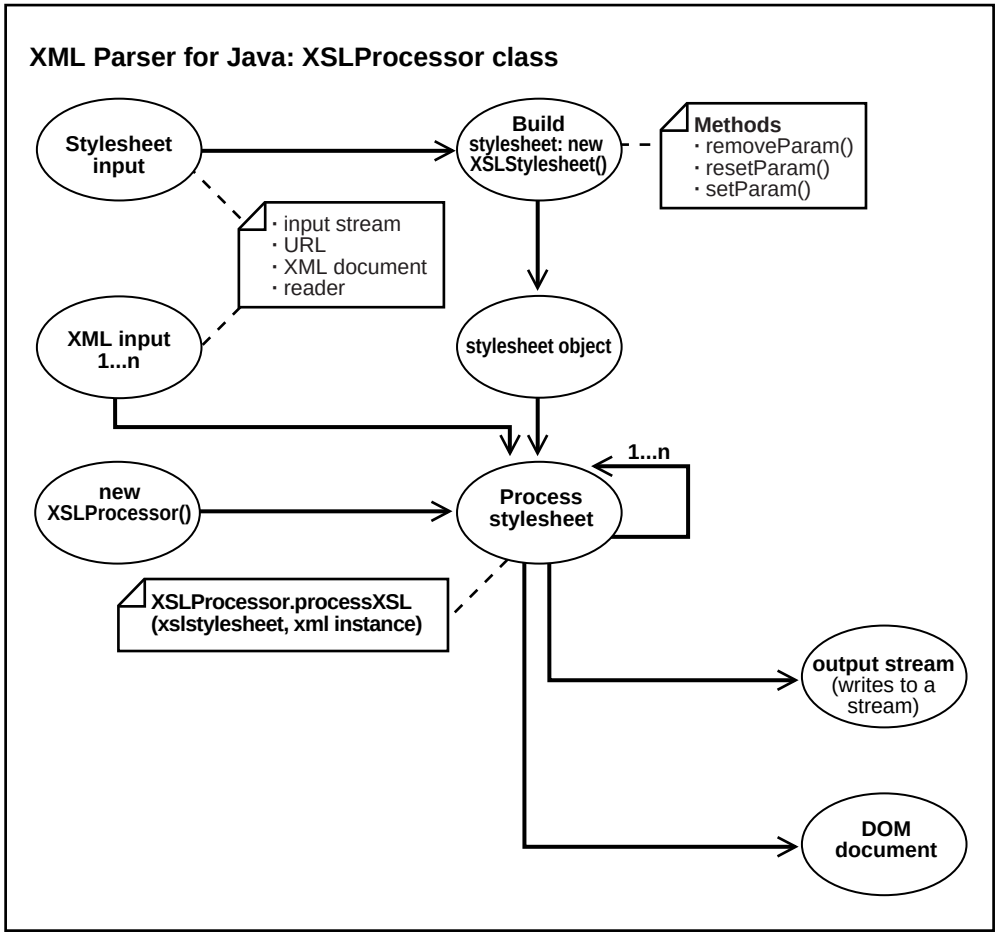
   ```
   XSLProcessor.processXSL(xslstylesheet, xml instance)
   ```

4. The XSLProcessor.processXSL() method processes the XML input 1 through n times, using the selected stylesheet.
5. XSLProcessor.processXSL() outputs either an output stream or a DOM document.

XML Parser for Java XSL-T Processor is illustrated by the following examples:

- "XML Parser for Java Example 4: (XSLSample.java)"
- "XML Parser for Java Example 5: Using the DOMAPI and XSL-T Processor"

Figure 17–6 XSLProcessor Class Process



XML Parser for Java Example 4: (XSLSample.java)

```
/**
 * This file gives a simple example of how to use the XSL processing
 * capabilities of the Oracle XML Parser V2.0. An input XML document is
 * transformed using a given input stylesheet
 */

import org.w3c.dom.*;
```

```
import java.util.*;
import java.io.*;
import java.net.*;
import oracle.xml.parser.v2.*;

public class XSLSample
{
   /**
    * Transforms an xml document using a stylesheet
    * @param args input xml and xml documents
    */
   public static void main (String args[]) throws Exception
   {
      DOMParser parser;
      XMLDocument xml, xsldoc, out;
      URL xslURL;
      URL xmlURL;

      try
      {

         if (args.length != 2)
         {
            // Must pass in the names of the XSL and XML files
            System.err.println("Usage: java XSLSample xslfile xmlfile");
            System.exit(1);
         }

         // Parse xsl and xml documents

         parser = new DOMParser();
         parser.setPreserveWhitespace(true);

         // parser input XSL file
         xslURL = createURL(args[0]);
         parser.parse(xslURL);
         xsldoc = parser.getDocument();

         // parser input XML file
         xmlURL = createURL(args[1]);
         parser.parse(xmlURL);
         xml = parser.getDocument();

         // instantiate a stylesheet
         XSLStylesheet xsl = new XSLStylesheet(xsldoc, xslURL);
```

```
      XSLProcessor processor = new XSLProcessor();

      // display any warnings that may occur
      processor.showWarnings(true);
      processor.setErrorStream(System.err);

      // Process XSL
      DocumentFragment result = processor.processXSL(xsl, xml);

      // create an output document to hold the result
      out = new XMLDocument();

      // create a dummy document element for the output document
      Element root = out.createElement("root");
      out.appendChild(root);

      // append the transformed tree to the dummy document element
      root.appendChild(result);

      // print the transformed document
      out.print(System.out);
   }
   catch (Exception e)
   {
      e.printStackTrace();
   }
}

// Helper method to create a URL from a file name
static URL createURL(String fileName)
{
   URL url = null;
   try
   {
      url = new URL(fileName);
   }
   catch (MalformedURLException ex)
   {
      File f = new File(fileName);
      try
      {
         String path = f.getAbsolutePath();
         // This is a bunch of weird code that is required to
         // make a valid URL on the Windows platform, due
         // to inconsistencies in what getAbsolutePath returns.
```

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

XML Parser for Java Example 5: Using the DOMAPI and XSL-T Processor

This example code is not included in the /sample subdirectory. The following Java code uses the XML Parser for Java, V2, to perform the following tasks.

- Parse an XML document
- Use the DOM API, to manipulate the XML data
- Use the XSL-T Processor to transform the data

```
import org.w3c.dom.*;
import java.util.*;
import java.io.*;
import java.net.*;
import oracle.xml.parser.v2.*;
public class XSLTransform
{
   public static void main (String args[]) throws Exception
   {
      DOMParser parser;
      XMLDocument xml, xsldoc, out;
```

```
URL xslURL;
URL xmlURL;

try
{
   if (args.length != 2)
   {
// Pass in the names of the XSL and XML files
   System.err.println("Usage: java XSLTransform
      xslfile xmlfile");
   System.exit(1);
   }

// Parse XSL and XML documents
   parser = new DOMParser();
   parser.setPreserveWhitespace(true);

   xslURL = createURL(args[0]);
   parser.parse(xslURL);
   xsldoc = parser.getDocument();

   xmlURL = createURL(args[1]);
   parser.parse(xmlURL);
   xml = parser.getDocument();

// Instantiate the stylesheet
   XSLStylesheet xsl = new XSLStylesheet(xsldoc, xslURL);

   XSLProcessor processor = new XSLProcessor();

// Display any warnings that may occur
   processor.showWarnings(true);
   processor.setErrorStream(System.err);

// Process XSL
   DocumentFragment result = processor.processXSL(xsl, xml);

// Create an output document to hold the result
   out = new XMLDocument();

// Create a dummy document element for the output document
   Element root = out.createElement("root");
   out.appendChild(root);

// Append the transformed tree to the dummy document element
```

```
            root.appendChild(result);

         // Print the transformed document
            out.print(System.out);
         }
         catch (Exception e)
         {
            e.printStackTrace();
         }
      }

}
```

Comments on XSL-T Example 5

See Figure 17–4 and Figure 17–6. The following provides comments for Example 5:

1. The program inputs two URL documents:
   - URL xmlURL;
   - URL xslURL;
2. Parse the two documents and set the preserve white space property:

   ```
   parser = new DOMParser();
   parser.setPreserveWhitespace(true);
   ```

3. Get the XSL and XML documenst

   ```
   xslURL = createURL(args[0]);
   parser.parse(xslURL);
   xsldoc = parser.getDocument();

   xmlURL = createURL(args[1]);
   xmlURL = createURL(args[1]);
   parser.parse(xmlURL);
   xml = parser.getDocument();
   ```

4. Initialize a new XSLStylesheet and XSLProcessor class:

   ```
   XSLStylesheet xsl = new XSLStylesheet(xsldoc, xslURL);

   XSLProcessor processor = new XSLProcessor();
   processor.setErrorStream(System.err);
   ```

5. Process the stylesheet

```
DocumentFragment result = processor.processXSL(xsl, xml);
```

6. Output the DOM XML transformed document

```
out = new XMLDocument();
Element root = out.createElement("root");
out.appendChild(root);
root.appendChild(result);
```

Using XML Parser for Java: SAXNamespace() Class

Using the SAXNamespace() class is illustrated in the following example:

XML Parser for Java Example 6: (SAXNamespace.java)

```
// This file demonstrates a simple use of the Namespace extensions to
// the SAX APIs.

import org.xml.sax.*;
import java.io.*;
import java.net.URL;
import java.net.MalformedURLException;

// Extensions to the SAX Interfaces for Namespace support.
import oracle.xml.parser.v2.XMLDocumentHandler;
import oracle.xml.parser.v2.DefaultXMLDocumentHandler;
import oracle.xml.parser.v2.NSName;
import oracle.xml.parser.v2.SAXAttrList;

import oracle.xml.parser.v2.SAXParser;

public class SAXNamespace {
  static public void main(String[] args) {
     String fileName;

     //Get the file name
     if (args.length == 0)
     {
        System.err.println("No file Specified!!!");
        System.err.println("USAGE: java SAXNamespace <filename>");
        return;
     }
     else
     {
        fileName = args[0];
     }

     try {
        // Create handlers for the parser
        // Use the XMLDocumentHandler interface for namespace support
        // instead of org.xml.sax.DocumentHandler
        XMLDocumentHandler xmlDocHandler = new XMLDocumentHandlerImpl();
```

```
        // For all the other interface use the default provided by
        // Handler base
        HandlerBase defHandler = new HandlerBase();

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

        // Set Handlers in the parser
        // Set the DocumentHandler to XMLDocumentHandler
        parser.setDocumentHandler(xmlDocHandler);

        // Set the other Handler to the defHandler
        parser.setErrorHandler(defHandler);
        parser.setEntityResolver(defHandler);
        parser.setDTDHandler(defHandler);

        try
        {
           parser.parse(fileToURL(new File(fileName)).toString());
        }
        catch (SAXParseException e)
        {
           System.err.println(args[0] + ": " + e.getMessage());
        }
        catch (SAXException e)
        {
           System.err.println(args[0] + ": " + e.getMessage());
        }
     }
     catch (Exception e)
     {
        System.err.println(e.toString());
     }
  }

static public URL fileToURL(File file)
   {
    String path = file.getAbsolutePath();
    String fSep = System.getProperty("file.separator");
    if (fSep != null && fSep.length() == 1)
      path = path.replace(fSep.charAt(0), '/');
    if (path.length() > 0 && path.charAt(0) != '/')
      path = '/' + path;
    try {
```

```
      return new URL("file", null, path);
     }
     catch (java.net.MalformedURLException e) {
       /* According to the spec this could only happen if the file
  protocol were not recognized. */
       throw new Error("unexpected MalformedURLException");
     }
   }

   private SAXNamespace() throws IOException
    {
    }

}
   /**************************************************************************
     Implementation of XMLDocumentHandler interface. Only the new
     startElement and endElement interfaces are implemented here. All other
     interfaces are implemented in the class HandlerBase.
     **************************************************************************/

class XMLDocumentHandlerImpl extends DefaultXMLDocumentHandler
{

   public void XMLDocumentHandlerImpl()
   {
   }


   public void startElement(NSName name, SAXAttrList atts) throws SAXException
   {

      // Use the methods getQualifiedName(), getLocalName(), getNamespace()
      // and getExpandedName() in NSName interface to get Namespace
      // information.
      String qName;
      String localName;
      String nsName;
      String expName;
      qName = name.getQualifiedName();
      System.out.println("ELEMENT Qualified Name:" + qName);
      localName = name.getLocalName();
      System.out.println("ELEMENT Local Name    :" + localName);

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

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

        for (int i=0; i<atts.getLength(); i++)
        {

        // Use the methods getQualifiedName(), getLocalName(), getNamespace()
        // and getExpandedName() in SAXAttrList interface to get Namespace
        // information.
           qName = atts.getQualifiedName(i);
           localName = atts.getLocalName(i);
           nsName = atts.getNamespace(i);
           expName = atts.getExpandedName(i);

           System.out.println(" ATTRIBUTE Qualified Name   :" + qName);
           System.out.println(" ATTRIBUTE Local Name       :" + localName);
           System.out.println(" ATTRIBUTE Namespace        :" + nsName);
           System.out.println(" ATTRIBUTE Expanded Name    :" + expName);

           // You can get the type and value of the attributes either
           // by index or by the Qualified Name.
           String type = atts.getType(qName);
           String value = atts.getValue(qName);

           System.out.println(" ATTRIBUTE Type             :" + type);
           System.out.println(" ATTRIBUTE Value            :" + value);
           System.out.println();
        }
     }

   public void endElement(NSName name) throws SAXException
     {
        // Use the methods getQualifiedName(), getLocalName(), getNamespace()
        // and getExpandedName() in NSName interface to get Namespace
        // information.
        String expName = name.getExpandedName();
        System.out.println("ELEMENT Expanded Name  :" + expName);
     }
}
```

XML Parser for Java: Command Line Interfaces

oraxml - Oracle XML parser

`oraxml` is a command-line interface to parse an XML document and check for well-formedness and validity.

To use oraxml ensure the following:

- Your CLASSPATH environment variable is set to point to the xmlparserv2.jar file that comes with Oracle XML V2 parser for Java.
- Your PATH environment variable can find the java interpreter that comes with JDK 1.1.x or JDK 1.2.

Use the following syntax to invoke `oraxml`:

```
oraxml options* source
```

`oraxml` expects to be given an XML file to parse. The options are described inTable 17–2.

Table 17–2 oraxml: Command Line Options

Option	Purpose
-h	Help mode (prints oraxml invocation syntax)
-v	Validation mode (if this option is not used, the parse check only for well formedness) -wShow warnings (by default, warnings are turned off)
-debug	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).

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 invole `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 17–3.

Table 17–3 oraxsl: Command Line Options

Option	Purpose
-h	Help mode (prints oraxsl invocation syntax)
-v	Verbose mode (some debugging information is printed and could help in tracing any problems that are encountered during processing)
-w	Show warnings (by default, warnings are turned off)
-debug	New - Debug mode (by default, debug mode is turned off)
-e <error log>	A file to write errors to (specify a log file to write errors and warnings).
-t <# of threads>	Number of threads to use for processing (using multiple threads could provide performance improvements when processing multiple documents).
-l <xml file list>	List of files to transform (allows you to explicitly list the files to be processed).
-d <directory>	Directory with files to transform (the default behavior is to process all files in the directory). If only a certain subset of the files in that directory, e.g., one file, need to be processed, this behavior must be changed by using -l and specifying just the files that need to be processed. You could also change the behavior by using the '-x' or '-i' option to select files based on their extension).
-x <source extension>	Extensions to exclude (used in conjunction with -d. All files with the specified extension will not be selected).
-i <source extension>	Extensions to include (used in conjunction with -d. Only files with the specified extension will be selected).

Table 17–3 oraxsl: Command Line Options (Cont.)

Option	Purpose
-s <stylesheet>	Stylesheet to use (if -d or -l is specified, this option needs to be specified to specify the stylesheet to be used. The complete path must be specified).
-r <result extension>	Extension to use for results (if -d or -l is specified, this option must be specified to specify the extension to be used for the results of the transformation. So, if one specifies the extension "out", an input document "foo" would get transformed to "foo.out". By default, the results are placed in the current directory. This is can be changed by using the -o option which allows you to specify a directory to hold the results).
-o <result directory>	Directory to place results (this must be used in conjunction with the -r option).

XML Extension Functions for XSL-T Processing

XSL-T Processor Extension Functions: Introduction

XML extension functions for XSL-T processing allow users of XSL-T processor to call any Java method from XSL expressions.

Java extension functions should belong to the namespace that starts with the following:

```
http://www.oracle.com/XSL/Transform/java/
```

An extension function that belongs to the following namespace:

```
http://www.oracle.com/XSL/Transform/java/classname
```

refers to methods in class `classname`.

For example, the following namespace:

```
http://www.oracle.com/XSL/Transform/java/java.lang.String
```

can be used to call `java.lang.String` methods from XSL expressions.

Static Versus Non-static Methods

If the method is a non-static method of the class, then the first parameter will be used as the instance on which the method is invoked, and the rest of the parameters are passed on to the method.

If the extension function is a static method, then all the parameters of the extension function are passed on as parameters to the static function.

XML Parser for Java - XSL Example 1: Static function

The following XSL, static function example:

```
<xsl:stylesheet
xmlns:math="http://www.oracle.com/XSL/Transform/java/java.lang.Math">
  <xsl:template match="/">
  <xsl:value-of select="math:ceil('12.34')"/>
</xsl:template>
</xsl:stylesheet>
```

prints out '13'.

Constructor Extension Function

The extension function 'new' creates a new instance of the class and acts as the constructor.

XML Parser for Java - XSL Example 2: Constructor Extension Function

The following constructor function example:

```
<xsl:stylesheet
xmlns:jstring="http://www.oracle.com/XSL/Transform/java/java.lang.String">
  <xsl:template match="/">
  <!-- creates a new java.lang.String and stores it in the variable str1 -->
  <xsl:variable name="str1" select="jstring:new('Hello World')"/>
  <xsl:value-of select="jstring:toUpperCase($str1)"/>
</xsl:template>
</xsl:stylesheet>
```

prints out 'HELLO WORLD'.

Return Value Extension Function

The result of an extension function can be of any type, including the five types defined in XSL:

- NodeList
- boolean
- String
- Number
- resulttree

They can be stored in variables or passed onto other extension functions.

If the result is of one of the five types defined in XSL, then the result can be returned as the result of an XSL expression.

XML Parser for Java XSL 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 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
```

Frequently Asked Questions (FAQs): XML Parser for Java

The XML Parser for Java Frequently Asked Questions (FAQs) are organized into the following topics:

DTDs

Checking DTD Syntax: Suggestions for Editors

Question

I was wondering if someone could help me verify the syntax for the following DTD. I realize that I can use a DTD editor to do this for me, but the editor I'm using is not very good.

```
<?xml version="1.0"?>
<!DOCTYPE CATALOG [

<!ELEMENT CATALOG ( ADMIN, SCHEMA?, DATA? ) >
<!ATTLIST CATALOG xml:lang NMTOKEN #IMPLIED >

<!ELEMENT ADMIN ( NAME, INFORMATION) >
<!ELEMENT SCHEMA (CATEGORY | DESCRIPTOR)* >
<!ELEMENT DATA (ITEM)*>
```

```
<!ELEMENT NAME (#PCDATA) >
<!ELEMENT INFORMATION ( DATE, SOURCE )  >
<!ELEMENT DATE (#PCDATA) >
<!ELEMENT SOURCE (#PCDATA) >

<!ELEMENT CATEGORY (NAME | KEY | TYPE | UPDATE  )* >
<!ATTLIST CATEGORY ACTION (ADD|DELETE|UPDATE) #REQUIRED>
<!ELEMENT DESCRIPTOR (NAME | KEY | UPDATE | OWNER  | TYPE )* >
<!ATTLIST DESCRIPTOR ACTION (ADD|DELETE|UPDATE) #REQUIRED>
<!ELEMENT OWNER (NAME?, KEY? ) >
<!ELEMENT KEY (#PCDATA) >
<!ELEMENT TYPE (#PCDATA) >

<!ELEMENT ITEM (OWNER?, NAMEVALUE*, UPDATE ) >
<!ATTLIST ITEM ACTION (ADD | DELETE | UPDATE) #REQUIRED>
<!ELEMENT UPDATE (NAME | KEY | NAMEVALUE )* >

<!ELEMENT NAMEVALUE ( NAME, VALUE ) >
<!ELEMENT VALUE (#PCDATA)* >
]>
```

I'm unsure about the ATTLIST syntax.

Answer

I loaded this into XMLAuthority 1.1 and did a Save As. XML Authority lets you visually inspect and edit DTD's and XML Schemas. Highly recommended. http://www.extensibility.com ($99.00).

It came back with:

```
<!ELEMENT CATALOG  (ADMIN , SCHEMA? , DATA? )>
<!ATTLIST CATALOG  xml:lang NMTOKEN  #IMPLIED >
<!ELEMENT ADMIN  (NAME , INFORMATION )>
<!ELEMENT SCHEMA  (CATEGORY | DESCRIPTOR )*>
<!ELEMENT DATA  (ITEM )*>
<!ELEMENT NAME  (#PCDATA )>
<!ELEMENT INFORMATION  (DATE , SOURCE )>
<!ELEMENT DATE  (#PCDATA )>
<!ELEMENT SOURCE  (#PCDATA )>
<!ELEMENT CATEGORY  (NAME | KEY | TYPE | UPDATE )*>
<!ATTLIST CATEGORY  ACTION  (ADD | DELETE | UPDATE )  #REQUIRED >
<!ELEMENT DESCRIPTOR  (NAME | KEY | UPDATE | OWNER | TYPE )*>
<!ATTLIST DESCRIPTOR  ACTION  (ADD | DELETE | UPDATE )  #REQUIRED >
<!ELEMENT OWNER  (NAME? , KEY? )>
```

```
<!ELEMENT KEY  (#PCDATA )>
<!ELEMENT TYPE  (#PCDATA )>
<!ELEMENT ITEM  (OWNER? , NAMEVALUE* , UPDATE )>
<!ATTLIST ITEM  ACTION  (ADD | DELETE | UPDATE )  #REQUIRED >
<!ELEMENT UPDATE  (NAME | KEY | NAMEVALUE )*>
<!ELEMENT NAMEVALUE  (NAME , VALUE )>
<!ELEMENT VALUE  (#PCDATA )*>
```

DTD File in DOCTYPE Must be Relative to XML Document Location

Question

My parser doesn't find the DTD file.

Answer

The DTD file defined in the <!DOCTYPE> declaration must be relative to the location of the input XML document. Otherwise, you'll need to use the setBaseURL(url) functions to set the base URL to resolve the relative address of the DTD if the input is coming from an InputStream.

Validating an XML File Using External DTD

Question

Can I validate an XML file using an external DTD?

Answer

You need to include a reference to the applicable DTD in your XML document. Without it there is no way that the parser knows what to validate against. Including the reference is the XML standard way of specifying an external DTD. Otherwise you need to embed the DTD in your XML Document.

DTD Caching

Question

Do you have DTD caching? How do I set the DTD using v2 parser for DTD Cache purpose?

Answer

Yes, DTD caching is optional and is not enabled automatically.

The method to set the DTD is setDoctype(). Here is an example:

```
// Test using InputSource
parser = new DOMParser();
parser.setErrorStream(System.out);
parser.showWarnings(true);

FileReader r = new FileReader(args[0]);
InputSource inSource = new InputSource(r);
inSource.setSystemId(createURL(args[0]).toString());
parser.parseDTD(inSource, args[1]);
dtd = (DTD)parser.getDoctype();

r = new FileReader(args[2]);
inSource = new InputSource(r);
inSource.setSystemId(createURL(args[2]).toString());
parser.setDoctype(dtd);
parser.setValidationMode(true);
parser.parse(inSource);

doc = (XMLDocument)parser.getDocument();
doc.print(new PrintWriter(System.out));
```

Recognizing External DTDs

Question

How can XML Parser for Java (V2) recognize external DTD's when running from the server. The Java code has been loaded with loadjava and runs in the Oracle8i server process. My XML file has an external DTD reference.

1. But is there a more generic way, as with the SAX parser, to redirect it to a stream or string or something if my DTD is in the database?
2. Do you have a more generic way to redirect the DTD, analogous to that offered by the SAXParser with resolveEntity().

Answer

1. We only have the setBaseURL() method at this time.
2. You can achieve your desired result using the following:

a. Parse your External DTD using a DOMParser's parseDTD() method.

b. Call getDoctype() to get an instance of oracle.xml.parser.v2.DTD

c. On the document where you want to set your DTD programmatically, use the: setDoctype(yourDTD); We use this technique to read a DTD out of our product's JAR file..

Loading external DTD's from a jar File

Question

I would like to put all my DTDs in a jar file, so then when the XML Parser needs a DTD it can get it from the jar. The XML Parser right now supports a base URL(setBaseURL()), but that just points to a place where all the DTDs are exposed.

Answer

The solution involves a combination of:

1. Load DTD as InputStream using:

```
InputStream is =
YourClass.class.getResourceAsStream("/foo/bar/your.dtd");
```

This will open ./foo/bar/your.dtd in the first relative location on the CLASSPATH that it can be found, including out of your jar if it's in the CLASSPATH.

2. Parse the DTD with the code:

```
DOMParser d = new DOMParser();
d.parseDTD(is, "rootelementname");
d.setDoctype(d.getDoctype());
```

3. Now parse your document with:

```
d.parse("yourdoc");
```

Can I Check the Correctness of an XML Document Using their DTD?

Question

I am exporting Java objects to XML. I can construct a DOM with a XMLDocument and use its print method to export it. But, I am unable to set the DTD of these

documents. I construct a parser, parse the DTD, and then get the DTD viaDocument doc = parser.getDocument() andDocType dtd = doc.getDocumentType().

How do I set the DTD of the freshly constructed XMLDocuments to use this one in order to be able to check the correctness of the documents using this DTD at a later time?

Answer

Your method of getting the DTD object is correct. However, we do not do any validation while creating the DOM tree using DOM APIs. So setting the DTD in the Document will not help validate the DOM tree that is constructed. The only way to validate an XML file is to parse the XML document using DOMParser or SAXParser.

Parsing a DTD Object Separately from XML Document

Question

How do I parse and get a DTD Object seperately from parsing my XML document?

Answer

The parseDTD() method allows you to parse a DTD file separately and get a DTD object. Here is a sample code to do that:

```
DOMParser domparser = new DOMParser();
domparser.setValidationMode(true);
/* parse the DTD file */
domparser.parseDTD(new FileReader(dtdfile));
DTD dtd = domparser.getDocType();
```

Case-Sensitivity in Parser Validation against DTD?

Question

The XML file has a tag like : <xn:subjectcode>. In the DTD, it is defined as `<xn:subjectCode>`. When the file is parsed and validated against the DTD, it gives an error: XML-0148 : (Error) Invalid element 'xn:subjectcode' in content of 'xn:Resource', ...

When I changed the element name to <xn:subjectCode> instead of <xn:subjectcode> it works. Is the parser case-sensitive as far as validation against DTD's go - or is it

because, there is a namespace also in the tag definition of the element and when a element is defined along with its namespace, the case-sensitivity comes into effect?

Answer

XML is inherently case-sensitive, therefore our parsers enforce case sensitivity in order to be compliant. When you run in non-validation mode only well-formness counts. However <test></Test> would signal an error even in non-validation mode.

Extracting Embedded XML From a CDATA Section

Question

1. I want to extract PAYLOAD and do extra processing on it.
2. When I select the value of PAYLOAD it does not parse the data because it is in a CDATA section.
3. How do I extract embedded XML using just XSLT. I have done this using SAX before but in the current setup all I can use is XSLT.

Answer

1. Here are the answers:

```
<PAYLOAD>
<![CDATA[<?xml version = '1.0' encoding = 'ASCII' standalone = 'no'?>
<ADD_PO_003>
   <CNTROLAREA>
      <BSR>
         <VERB value="ADD">ADD</VERB>
         <NOUN value="PO">PO</NOUN>
         <REVISION value="003">003</REVISION>
      </BSR>
   </CNTROLAREA>
</ADD_PO_003>]]>
</PAYLOAD>
```

   The CDATA strategy is kind of odd. You won't be able to use a different encoding on the nested XML document included as text inside the CDATA, so having the XML Declaration of the embedded document seems of little value to me. If you don't need the XML Declaration, then why not just embded the

message as real elements into the <PAYLOAD> instead of as a text chunk which is what CDATA does for you.

Just do:

```
String s = YourDocumentObject.selectSingleNode("/OES_MESSAGE/PAYLOAD");
```

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's nothing special about what's in your CDATA, it's just text. If you want the text content to be output without escaping the angle-brackets, then you'll do:

   ```
   <xsl:value-of select="/OES_MESSAGE/PAYLOAD" disable-output-escaping="yes"/>
   ```

DOM and SAX APIs

Using the DOM API

Question

How do I get the number of elements in a particular tag using the parser?

Answer

You can use the getElementsByTagName() method that returns a NodeList of all descent elements with a given tag name. You can then find out the number of elements in that NodeList to determine the number of the elements in the particular tag.

How DOM Parser Works

Question

How does the XML DOM parser work?

Answer

The parser accepts an XML formatted document and constructs in memory a DOM tree based on its structure. It will then check whether the document is well-formed and optionally whether it complies with a DTD. It also provides methods to support DOM Level 1.

Creating a Node With Value to be Set Later

Question

How do I create a node whose value I can set later?

Answer

If you check the DOM spec referring to the table discussing the node type, you will find that if you are creating an element node, its nodeValue is to be null and hence cannot be set. However, you can create a text node and append it to the element node. You can put the value in the text node.

Traversing the XML Tree

Question

How to traverse the XML tree

Answer

You can traverse the tree by using the DOM API. Or alternately, you can use the selectNodes() method which takes XPath syntax to navigate through the XML document. selectNodes() is part of oracle.xml.parser.v2.XMLNode.

Extracting Elements from XML File

Question

How do I extract elements from the XML file?

Answer

If you're using DOM, the getElementsByTagName() method can be used to get all of the elements in the document.

Does a DTD Validate the DOM Tree?

Question

If I add a DTD to an XML Document, does it validate the DOM tree?

Answer

No, we do not do any validation while creating the DOM tree using the DOM APIs. So setting the DTD in the Document will not help in validating the DOM tree that is constructed. The only way to validate an XML file is to parse the XML document using the DOMParser or SAXParser.

First Child Node Element Value

Question

How do I efficiently obtain the value of first child node of the element without going through the DOM Tree?

Answer

If you do not need the entire tree, use the SAX interface to return the desired data. Since it is event-driven, it does not have to parse the whole document.

Creating DocType Node

Question

How do I create a DocType Node?

Answer

The only current way of creating a doctype node is by using the parseDTD functions. For example, emp.dtd has the following DTD:

```
<!ELEMENT employee (Name, Dept, Title)>
 <!ELEMENT Name (#PCDATA)>
<!ELEMENT Dept (#PCDATA)>
 <!ELEMENT Title (#PCDATA)>
```

You can use the following code to create a doctype node:

```
parser.parseDTD(new FileInputStream(emp.dtd), "employee");
dtd = parser.getDocType();
```

XMLNode.selectNodes() Method

Question

How do I use the selectNodes() method in XMLNode class?

Answer

The selectNodes() method is used in XMLElement and XMLDocument nodes. This method is used to extract contents from the tree / subtree based on the select patterns allowed by XSL. The optional second parameter of selectNodes, is used to resolve Namespace prefixes (return the expanded namespace URL given a prefix). XMLElement implements NSResolver, so it can be sent as the second parameter. XMLElement resolves the prefixes based on the input document. You can implement the NSResolver interface, if you need to overide the namespace definitions. The following sample code uses selectNodes

```
public class SelectNodesTest  {
```

```
public static void main(String[] args) throws Exception {
String pattern = "/family/member/text()";
String file    = args[0];

if (args.length == 2)
  pattern = args[1];

DOMParser dp = new DOMParser();

dp.parse(createURL(file));  // Include createURL from DOMSample
XMLDocument xd = dp.getDocument();
XMLElement e = (XMLElement) xd.getDocumentElement();
NodeList nl = e.selectNodes(pattern, e);
for (int i = 0; i < nl.getLength(); i++) {
   System.out.println(nl.item(i).getNodeValue());
    }
  }
}

> java SelectNodesTest family.xml
Sarah
Bob
Joanne
Jim

> java SelectNodesTest family.xml //member/@memberid
m1
m2
m3
m4
```

Using SAX API to Get the Data Value

Question

I am using SAX to parse an XML document. How does it get the value of the data?

Answer

During a SAX parse the value of an element will be the concatenation of the characters reported from after the startElement event to before the corresponding endElement event is called.

SAXSample.java

Question

Inside the SAXSample program, I did not see any line that explicitly calls setDocumentLocator and some other methods. However, these methods are 'run'. Can you explain when they are called and from where

Answer

SAX is a standard interface for event-based XML parsing. The parser reports parsing events directly through callback functions such as setDocumentLocator() and startDocument(). The application, in this case, the SAXSample, implements handlers to deal with the different events. Here is a good place to help you start learning about the event-driven API, SAX:
http://www.megginson.com/SAX/index.html

Does DOMParser implement Parser interface

Question

Does the XML Parser DOMParser implement org.xml.sax.Parser interface at all ? The documentation says it implements XMLConstants and the API does not include that class at all.

Answer

You'll want oracle.xml.parser.v2.SAXParser to work with SAX and to have something that implements the org.xml.sax.Parser interface.

Creating an New Document Type Node Via DOM

Question

I am trying to create a XML file on the fly. I use the NodeFactory to contruct a document (createDocument()). I have then setStandalone("no") and setVersion("1.0"). when I try to add a DOCTYPE node via appendChild(new XMLNode("test", Node.DOCUMENT_TYPE_NODE)), I get a ClassCastException. What is the mechanism to add a node of this type? I noticed that the NodeFactory did not have a machanism for creating a DOCTYPE node.

Answer

There is no mechanism to create a new DOCUMENT_TYPE_NODE object via DOM APIs. The only way to get a DTD object is to parse theDTD file or the XML file using the DOMParser, and then use the getDocType() method.

Note that new XMLNode("test",Node.DOCUMENT_TYPE_NODE) does not create a DTDobject. It creates an XMLNode object with the type set toDOCUMENT_ TYPE_NODE, which infact should not be allowed. TheClassCastException is raised because appendChild expects a DTDobject (based on the type).

Also, we do not do any validation while creating the DOM tree using the DOM APIs. So setting the DTD in the Document will not help in validating the DOM tree that is constructed. The only way to validate an XML file is to parse the XML document using DOMParser or SAXParser.

Querying for First Child Node's Value of a Certain Tag

Question

I am using the XML Parser for Java v2. Given a XML document containing the following Calculus Math Jim Green Jack Mary Paul, I want to obtain the value of first child node of whose tag is . I could not find any method that can do that efficiently. The nearest match is method 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.

XML Document Generation From Data in Variables

Question

Is there an example of XML document generation starting from information contained in simple variables? An example would be: A client fills a Java form and wants to obtain an XML document containing the given data.

Answer

Here are two possible interpretations of your question and answers to both. Let's say you have two variables in Java:

```
String firstname = "Gianfranco";
String lastname = "Pietraforte";
```

The two ways that come to mind first to get this into an XML document are as follows:

1. Make an XML document in a string and parse it.

```
String xml = "<person><first>"+firstname+"</first>"+
        "<last>"+lastname+"</last></person";

 DOMParser d = new DOMParser();
 d.parse( new StringReader(xml));
 Document xmldoc = d.getDocument();
```

2. Use DOM API's to construct the document and "stitch" it together:

```
Document xmldoc = new XMLDocument();
Element e1 = xmldoc.createElement("person");
xmldoc.appendChild(e1);
Element e2 = xmldoc.createElement("first");
e1.appendChild(e2);
Text t = xmldoc.createText(firstname);
e2.appendChild(t);
// and so on
```

Printing Data in the Element Tags: DOM API

Question

Can you suggest how to get a print out using the DOM API in Java:

```
<name>macy</name>
```

I want to print out "macy". Dont know which class and what fucntion 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 followinig:

```
String value = myElement.getFirstChild().getNodeValue();
```

Building XML Files from Hashtable Value Pairs

Question

We have a hash table of key value pairs, how do we build an XML file out of it using the DOM API? We have a hashtablekey = valuename = georgezip = 20000. How do we build this?

```
<key>value</key><name>george</name><zip>20000</zip>'
```

Is there a utility to do it automatically?

Answer

1. Get the enumeration of keys from your hashtable
2. Loop while enum.hasMoreElements()
3. For each key in the enumeration, use the createElement() on DOM Document to create an element by the name of the key with a child text node with the value of the *value* of the hashtable entry for that key.

XML Parser for Java: wrong_document_err on Node.appendChild()

Question

I have a question regarding our XML parser (v2) implementation. Say if I have the following scenario:

```
Document doc1 = new XMLDocument();
Element element1 = doc1.creatElement("foo");
Document doc2 = new XMLDocument();
Element element2 = doc2.createElement("bar");
element1.appendChild(element2);
```

My question is whether or not we should get a DOMException of WRONG_ DOCUMENT_ERR on calling the appendChild() routine. This comes to my mind when I look at the XSLSample.java distributed with the XMLparser (v2). Any feedback would be greatly appreciated.

Answer

Yes you should get this error, since the owner document of element1 is doc1 while that of element2 is doc2. AppendChild() only works within a single tree and you are dealing with two different ones.

Question 2

In XSLSample.java that's shipped with xmlparser v2: DocumentFragment result = processor.processXSL(xsl, xml); / / create an output document to hold the result out = new XMLDocument(); / / create a dummy document element for the output document Element root = out.createElement("root"); out.appendChild(root); / / append the transformed tree to the dummy document element root.appendChild(result);Nodes root and result are created from different XMLDocuments, wouldn't this result in the WRONG_DOCUMENT_ERR when we try to append result to root?

Answer 2

This sample is using a document fragment which does not have a root node, therefore there are not two XML documents.

Question 3

When appending a document fragment to a node, only the child nodes of the document fragment (but not the document fragment itself) is inserted. Wouldn't the parser check the owner document of these child nodes? I believe that's the behavior of the IBM xml parser.

Comment

I am experimenting the same sort of problem. From my personal point of view, a DocumentFragment shouldn't be bound to a 'root' node in any case, since, by definition, a frangment could very well be just a list of nodes ... The root node, if any, should be considered a single child. i.e. you cold for example take all the lines of an Invoice document, and add them into an ProviderOrder document, without taking the invoice itself...Don't really know if this makes sense... anyway, how do we create a documentFragment without root?, same as the XSL-T Processor does, so that we can append it to other document?

Creating Nodes: DOMException when Setting Node Value

Question

I get the following error:

```
oracle.xml.parser.XMLDOMException: Node cannot be modifiedwhile trying to set
the value of a newly created node as below:     String eName="Mynode";
XMLNode aNode = new XMLNode(eName, Node.ELEMENT_NODE);
aNode.setNodeValue(eValue);
```

How do I create a node whose value I can set later on?

Answer

Check the DOM notes where they discuss the node type. You will see that if you are creating an element node, its nodeValue is null and hence cannot be set.

Validation

DTD: Understanding DOCTYPE and Validating Parser

Question

I have an XML string contains the following reference to a DTD, that is physically located in the directory where I start my program. The validating XML parser complains that this file can not be found.

```
<!DOCTYPE xyz SYSTEM "xyz.dtd" >
```

What are the rules for locating DTDs on the disk? Can anyone point me to a decent discussion of DOCTYPE attribute descriptions.

Answer

Are you parsing an InputStream oran URL ? If you are parsing an InputStream the parser doesn't knowwhere that InputStream came from so it cannot find the DTD in the "samedirectory as the current file". The solution is to setBaseURL()on DOMParser() to give the parser the URL "hint" information to be able to derive the rest when it goes to get the DTD.

Can Multiple Threads Use Single XSLProcessor/Stylesheet?

Question

Can multiple threads use a single XSLProcessor / XSLStylesheet instance to perform concurrent (at the same time) transformations?

Answer

As long as you are processing multiple files with no more than one XSLProcessor / XSLStylesheet instance per XML file you can do this simultaneously using threads. If you take a look at the readme.html file in the bin directory, it describes ORAXSL which has a threads parameter for multi-threaded processing.

Is it Safe to Use Document Clones in Multiple Threads?

Question

Is it safe to use clones of a document in multiple threads? Is the public void setParam(String ,String) throws XSLExceptionmethode of Class oracle.xml.parser.v2.XSLStylesheet supported? If no, is there another way to pass parameters at runtime to the XSL-T Processor?

Answer

If you are copying the global area set up by the constructor to another thread then it should work.

That method is supported since XML Parser release 2.0.2.5.

Comment

You have it in your docs, but it is not implemented in the XSLStylesheet class (windows zip edition). First update your zip download file.

```
public static void serve(Document template, Document data,Element
userdata,PrintWriter out)
{
 XMLDocument clone = (XMLDocument)data.cloneNode(true);
    clone.getDocumentElement().appendChild(userdata.cloneNode(true));
    serve(template, clone, out);
}
```

Character Sets

Encoding iso-8859-1 in xmlparser

Question

I have some XML-Documents with encoding="iso-8859-1". I am trying to parse these with xmlparser SAX API. In characters (char[], int, int), I would like to output the content in iso-8859-1 (Latin1) too.

With System.out.println() it doesn't work correctly. German umlauts result in '?' in the output stream. Internally „, ,', ,Ù,›,¸ ar stored as 65508,65526,65532,65476,65494,65500,65503 respectively. What do I have to do to get the output in Latin1? Host system here is a SPARC Solaris 2.6.

Answer

You cannot use System.out.println(). You need to use an output stream which is encoding aware, for example, OutputStreamWriter.

You canconstruct an outputstreamwrite and use the write(char[], int, int) method to:

```
print.Ex:OutputStreamWriter out = new OutputStreamWriter(System.out, "8859_1");
/* Java enc string for ISO8859-1*/
```

Parsing XML Stored in NCLOB With UTF-8 Encoding

Question

I'm having trouble with parsing XML stored in NCLOB column using UTF-8 encoding. Here is what I'm running:

- Windows NT 4.0 ServerOracle 8i (8.1.5)
- EEJDeveloper 3.0
- JDK 1.1.8
- Oracle XML Parser v2 (2.0.2.5?)

The following XML sample that I loaded into the database contains two UTF-8 multi-byte characters:

```
<?xml version="1.0" encoding="UTF-8"?>
<G>
<A>GÂ,otingen, BrÃ ck_W</A>
```

```
</G>
```
G(0xc2, 0x82)otingen, Br(0xc3, 0xbc)ck_W

If I'm not mistaken, both multibyte characters are valid UTF-8 encodings and they are defined in ISO-8859-1 as:

```
 0xC2 LATIN CAPITAL LETTER A WITH CIRCUMFLEX
0xFC  LATIN SMALL LETTER U WITH DIAERESIS
```

I wrote a Java stored function that uses the default connection object to connect to the database, runs a Select query, gets the OracleResultSet, calls the getCLOB method and calls the getAsciiStream() method on the CLOB object. Then it executes the following piece of code to get the XML into a DOM object:

```
DOMParser parser = new DOMParser();
parser.setPreserveWhitespace(true);
parser.parse(istr);
// istr getAsciiStreamXMLDocument xmldoc = parser.getDocument();
```

Before the stored function can do other tasks, this code throws an exception complaining that the above XML contains "Invalid UTF8 encoding".

- When I remove the first multibyte character (0xc2, 0x82) from the XML, it parses fine.
- When I do not remove this character, but connect via the JDBC racle:thin driver (note that now I'm not running inside the RDBMS as stored function anymore) the XML is parsed with no problem and I can do what ever I want with the XMLDocument.

I loaded the sample XML into the database using the thin JDBC driver. I tried two database configurations with WE8ISO8859P1/WE8ISO8859P1 and WE8ISO8859P1/UTF8 and both showed the same problem.

Answer

Yes, the character (0xc2, 0x82) is valid UTF-8. We suspect that the character is distorted when getAsciiStream() is called. Try to use getUnicodeStream() and getBinaryStream() instead of getAsciiStream().

If this does not work, try print out the characters before to make sure that they are not distorted before they are sent to the parser in step: parser.parse(istr)

NLS support within XML

Question

I've got Japanese data stored in an nvarchar2 field in the database. I have a dynamic SQL procedure that utilizes the PL/SQL web toolkit that allows me to access data via OAS and a browser. This procedure uses the XML Parser to correctly format the result set in XML before returning it to the browser.

My problem is that the Japanese data is returned and displayed on the browser as upside down question marks. Is there anything I can do so that this data is correctly returned and displayed as Kanji?

Answer

Unfortunately, Java and XML default character set is UTF8 while I haven't heard of any UTF8 OS nor people using it as in their database and people writing their web pages in UTF8. All this means is that you have a character code conversion problem. Answer to your last question is 'yes'. We do have both PL/SQL and Java XML parsers working in Japanese. Unfortunately, we cannot provide a simple solution that will fit in this space.

UTF-16 Encoding with XML Parser for Java V2

Question

This is my XML Document:

Documento de Prueba de gestin de contenidos. Roberto P‰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 V2_0_2_5 and I am confused because the documentation says that the UTF-16 encoding is supported in this version of the Parser. Does anybody know how can I parse documents containing spanishaccents?

Answer

Oracle just uploaded a new release of V2 Parser. It should support UTF-16.Yet, other utilities still have some problems with UTF-16encoding.

Adding XML Document as a Child

Adding an XMLDocument as a Child to Another Element

Question

I am trying to add an XMLDocument as a child to an existing element. Here's an example:

```
import org.w3c.dom.*;
import java.util.*;
import java.io.*;
import java.net.*;
import oracle.xml.parser.v2.*;
public class ggg {public static void main (String [] args) throws Exception
 {
new ggg().doWork();;
public void doWork() throws Exception {XMLDocument doc1 = new XMLDocument();
Element root1=doc1.createElement("root1");
XMLDocument doc2= new XMLDocument();Element root2=doc2.createElement("root2");
root1.appendChild(root2);
doc1.print(System.out);};};
```

This reports:

```
D:\Temp\Oracle\sample>c:\jdk1.2.2\bin\javac -classpath
D:\Temp\Oracle\lib\xmlparserv2.jar;.
ggg.javaD:\Temp\Oracle\sample>c:\jdk1.2.2\bin\java -classpath
D:\Temp\Oracle\lib\xmlparserv2.jar;. gggException in thread "main"
java.lang.NullPointerException        at
oracle.xml.parser.v2.XMLDOMException.(XMLDOMException.java:67)        at
oracle.xml.parser.v2.XMLNode.checkDocument(XMLNode.java:919)        at
oracle.xml.parser.v2.XMLNode.appendChild(XMLNode.java, Compiled Code)        at
oracle.xml.parser.v2.XMLNode.appendChild(XMLNode.java:494)        at
```

```
ggg.doWork(ggg.java:20)        at ggg.main(ggg.java:12)
```

Answer a

The following works for me: :

```
DocumentFragment rootNode = new XMLDocumentFragment(); DOMParser d  = new
DOMParser(); d.parse("http://.../stuff.xml");
Document doc = d.getDocument();
Element e = doc.getDocumentElement();
// Important to remove it from the first doc
// before adding it to the other doc. doc.removeChild(e);
rootNode.appendChild(e);
```

You need to use the DocumentFragment class to do this as a document cannot have more than one root.

Answer b

Actually, isn't this specifically a problem with appending a node created in another document, since all nodes contain a reference to the document they are created in? While Document Fragmentsolves this, it isn't a more than one root problem, is it? Is there a quick or easy way to convert a com.w3c.dom.Document to org.w3c.dom.DocumentFragment?

Adding an XMLDocumentFragment as a Child to XMLDocument

Question

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 Exception in thread "main" oracle.xml.parser.v2.

```
XMLDOMException: Node of this type cannot be added.
```

Do I have to create a root element _every time_, even if I know that the resulting DocumentFragment is a well formed XML Document (and of course has only one root element!)?

Answer

It happens, as you have guessed, because a Fragment can have more than one "root" element (for lack of a better term). In order towork around this, use the Node functions to extract the one rootelement from your fragment and cast it into an

Uninstalling Parsers

Removing XML Parser from the Database

Question

I am deinstalling a version of XML Parser and installing a newer version. How do I do that? I know that there is something like dropjava , but still there are other packages which are loaded into the schema. I want to clean out the earlier version and install the new version in a clean manner.

Answer

You'll need to write SQL to write SQLbased on the USER_OBJECTS table where:

```
SELECT 'drop java class '''&#0124; &#0124;       dbms_java.longname(object_
name)&#0124; &#0124;''';
from user_objects where
OBJECT_TYPE = 'JAVA CLASS'and DBMS_JAVA.LONGNAME(OBJECT_NAME)     LIKE
'oracle/xml/parser/%'
```

This will spew out a set of DROP JAVA CLASS command which you can capture in a file using SQL*Plus': SPOOL somefilenamecommand.

Then run that spool file as a SQL script and all the right classeswill be dropped.

XML Parser for Java: Installation

XMLPARSER Fails to Install

Question

I'm getting an error message when I try installing XMLPARSER:

```
loadjava -user username/manager -r -v xmlparserv2.jar
Error:
Exception in thread "main" java.lang.NoClassDefFounderr:
oracle/jdbc/driver/OracleDriver at oracle.aurora.server.tools.. etc..
```

Answer

This is a failure to find the JDBC classes111.zip in your classpath. The loadjava utility connects to the database to load your classes using the JDBC driver.

I checked 'loadjava' and the path to classes111.zip is

```
<ORACLE_HOME>/jdbc/lib/classes111.zip
```

In version 8.1.6, classes111.zip resides in:

```
<ORACLE_HOME/jdbc/admin
```

General XML Parser Related Questions

How the XML Parser Works

Question

What does an XML Parser do?

Answer

The parser accepts any XML document giving you a tree-based API (DOM) to access or modify the document's elements and attributes as well as an event-API (SAX) that provides a listener to be registered and report specific elements or attributes and other document events.

Converting XML Files to HTML Files

Question

How do I convert XML files into HTML files?

Answer

You need to create an XSL stylesheet to render your XML into HTML. You can start with an HTML document in your desired format and populated with dummy data. Then you can replace this data with the XSL-T commands that will populate the HTML with data from the XML document completing your stylesheet.

Validating Against XML Schema

Question

Does the XML Parser v2 validate against an XML Schema ?

Answer

It supports both validating and non-validating modes. XML Schema is still under the development W3C XML Schema committee. We will support it as soon as it becomes stable. Currently, our parser supports validating and non-validating DTDs.

Including Binary Data in an XML Document

Question

How do I include binary data in an XML document?

Answer

There is no way to directly include binary data within the document; however, there are two ways to work around this:

- Binary data could be referenced as an external unparsed entity that resided in a different file.
- Binary data can be uuencoded (meaning converting binary data into ASCII data) and be included in a CDATA section. The limitation on the encoding technique is to ensure that it only produces legal characters for the CDATA section.

What is XML Schema?

Question

What is the XML Schema?

Answer

XML Schema is a W3C XML standards effort to bring the concept of data types to XML documents and in the process replace the syntax of DTDs to one based on XML. For more details, check out http://www.w3.org/TR/xmlschema-1/ and http://www.w3.org/TR/xmlschema-2/.

Oracle's Participation in Defining the XML/SQL Standard

Question

Does Oracle participate in defining the XML/XSL standard?

Answer

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

XDK Version Numbers

Question

How do I determine the version number of the XDK toolkit that I downloaded?

Answer

You can find out the full version number by looking at the readme.html file included in the archive and linked off of the Release Notes page.

Inserting <, >, >= and <= in XML Documents

Question

How do I insert these characters in the XML documents: >,<,>=, and <=?

Answer

You need to use the entities < for < and > for >.

Are Namespace and Schema Supported

Question

Is support for Namespaces and Schema included?

Answer

The current XML parsers support Namespaces. Schema support will be included in a future release.

Using JDK 1.1.x with XML Parser for Java v2

Question

Can I use JDK 1.1.x with XML Parser v2 for Java?

Answer

v2 of XML Parser for Java has nothing to do with Java2. It is simply a designation that indicates that it is not backwards compatible with the v1 Parser and that it includes XSLT support. The v2 parser will work fine with JDK 1.1.x.

Sorting the Result on the Page

Question

I have a set of records say 100, I am showing 10 at a time, now on each column name I have made a link, on the click of the same, I want to sort the data in the page alone, based on that column. How to go about it?

Answer

It depends on how you are going about. If you are writing for IE5 alone and receiving XML data, you could just use MS's XSL to sort data in a page.If you are writing for other browser and browsers are getting data as HTML, then you have to have a sort parameter in XSQL script and use it in ORDER BY clause. Just passed it along with skip-rows parameter.

Is Oracle8*i* Needed to Run XML Parser for Java?

Question

Do I need Oracle8i to run the XML Parser for Java?

Answer

XML Parser for Java can be used with any of the supported version JavaVMs. The only difference with 8i is that you can load it into the database and use JServer, which is an internal JVM. For other database versions or servers, you simply run it in an external JVM and as necessary connect to a database through JDBC.

Dynamically Setting the Encoding in an XML File

Question

Is it possible to dynamically set the encodings in the XML file?

Answer

No, you need to include the proper encoding declaration in your document as per the specification. You cannot use setEncoding() to set the encoding for you input document. SetEncoding() is used with oracle.xml.parser.v2.XMLDocument to set the correct encoding for the printing.

Parsing a String

Question

How do I parse a string?

Answer

We do not currently have any method that can directly parse an XML document contained within a String. You would need to convert the String into an InputStream or InputSource before parsing. An easy way is to create a ByteArrayInputStream using the bytes in the String.

Displaying an XML Document

Question

How do I display my XML document?

Answer

If you are using IE5 as your browser you can display the XML document directly. Otherwise, you can use our XSL-T Processor in v2 of the parser to create the HTML document using an XSL Stylesheet. The Oracle XML Transviewer bean also allows you to view your XML document.

System.out.primtln() and Special Characters

Question

I am having problems using System.out.println() with special character encoding.

Answer

You can't use System.out.println(). You need to use an output stream which is encoding aware (Ex.OutputStreamWriter). You can construct an OutputStreamWriter and use the write(char[ ], int, int) method to print.

```
/* Example */
OutputStreamWriter out = new OutputStreamWriter
(System.out, "8859_1");
/* Java enc string for ISO8859-1*/
```

Obtaining Ampersand from Character Data

Question

How do I to get ampersand from character data?

Answer

You cannot have "raw" ampersands in XML data. You need to use the entity, & instead. This is defined in the XML standard.

Parsing XML from Data of Type String

Question

How do I parse XML from data of type String?

Answer

Check out the following example:

```
/* xmlDoc is a String of xml */
byte aByteArr [] = xmlDoc.getBytes();
ByteArrayInputStream bais = new ByteArrayInputStream (aByteArr, 0,
aByteArr.length);
domParser.parse(bais);
```

Extracting Data from XML Document into a String

Question

How do I extract data from an XML document into type String?

Answer

Here is an example to do that:

```
XMLDocument Your Document;
/* Parse and Make Mods */
:
StringWriter sw = new StringWriter();
PrintWriter  pw = new PrintWriter(sw);
YourDocument.print(pw);
String YourDocInString = sw.toString();
```

Disabling Output Escaping

Question

Does XML Parser for Java support Disabling Output Escaping?

Answer

Yes, since version 2.022, the parser provides an option to xsl:text to disable output escaping.

Using the XML Parser for Java with Oracle 8.0.5

Question

Is the XML Parser for Java only available for use with Oracle 8i? Is it possible to use with Oracle 8.05

Answer

The XML Parser for Java can be used with any of the supported version JavaVMs. The only difference with Oracle8*i* is that you can load it into the database and use JServer which is an internal VM. For 8.0.5 you simple run it externally and connect through JDBC.

Delimiting Multiple XML Documents

Question

We need to be able to read (and separate) several XML documents as a single string. One solution would be to delimit these documents using some (programatically

generated) special character that we know for sure can never occur inside an xml document. The individual documents can then be easily tokenized and extracted / parsed as required.

Has any one else done this before? Any suggestions for what character can be used as the delimiter (for instance can characters in the range #x0-#x8 ever occur inside an xml document?)

Answer

As far as legality is concerned and you limit it to 8-bit, #x0-#x8; #xB, #xC, #xE, and #xF are not legal. HOWEVER this assumes that you preprocess the doc and not depend upon exceptions as not ALL parsers reject ALL illegal characters.

Element, which you then append to the Document.

XML and Entity-references: XML Parser for Java

Question

1. The XML-parser for Java does not expand entity references,such as &[whatever], instead all values are null. How can I fix this?
2. It seems you cannot have international character (such as swedish characters, …„) as values for internal entities. How does one solve this problem?

Answer

1. You probably have a simple error defining / using your entities since we've a number of regression tests that handle entity references fine. A simple example is:]> Alpha, then &status
2. What do you set your character set encoding to be?

Can I Break up and Store an XML Document without a DDL Insert?

Question

1. We would like to break apart an arbitrary XML document and store it in the database without creating a DDL to insert. Is this possible?
2. And as for querying, is it possible to perform hierarchical searches across XML documents?

Answer

1. No this is not possible. Either the schema must already exist or and XSL stylesheet to create the DDL from the XML must exist.
2. From Oracle8*i* Release 8.1.6 *inter*Media Text can do this.

Merging XML Documents

Question

How can I merge two XML Documents?

Answer

This is not possible with the current DOM specification. DOM2 specification may address this.

You can use a DOM-approach or an XSL-T-based approach to accomplish this. If you use DOM, then you'll have to remove the node from one document before you append it into the other document to avoid ownership errors.

Here's an example of the XSL-based approach. Assume your two XML source files are:

demo1.xml

```
<messages>
  <msg>
    <key>AAA</key>
    <num>01001</num>
  </msg>
  <msg>
    <key>BBB</key>
    <num>01011</num>
  </msg>
</messages>
```

demo2.xml

```
<messages>
  <msg>
    <key>AAA</key>
    <text>This is a Message</text>
  </msg>
  <msg>
    <key>BBB</key>
```

```
    <text>This is another Message</text>
  </msg>
</messages>
```

Here is a stylesheet the "joins" demo1.xml to demo2.xml based on matching the "<key>" values.

demomerge.xsl

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output indent="yes"/>
<xsl:variable name="doc2" select="document('demo2.xml')"/>
  <xsl:template match="@*|node()">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
  </xsl:template>
<xsl:template match="msg">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()"/>
      <text><xsl:value-of select="$doc2/messages/msg[key=current()/key]/text"/>
</text>
    </xsl:copy>
</xsl:template>
</xsl:stylesheet>
```

If you use the command-line "`oraxsl`" to test this out, you would do:

```
$ oraxsl demo1.xml demomerge.xsl
```

And you'll get the merged result of:

```
<messages>
  <msg>
    <key>AAA</key>
    <num>01001</num>
    <text>This is a Message</text>
  </msg>
  <msg>
    <key>BBB</key>
    <num>01011</num>
    <text>This is another Message</text>
  </msg></messages>
```

Obviously not as efficient for larger-sized files as an equivalent database "join" between two tables, but this illustrates the technique if you only have XML files to work with.Error: Cannot Find Class

Getting the Value of a Tag

Question

I am using SAX to parse an XML document. How I can get the value of a particular tag? For example, Java. How do I get the value for title? I know there are startElement, endElement, and characters methods.

Answer

During a SAX parse the *value* of an element will be the concatenation of the characters reported from after startElement to before the corresponding endElement is called.

Granting JAVASYSPRIV to User

Question

We are using Oracle XML Parser for Java on NT 4.0. When we are parsing an XML document with an external DTD we get the following error:

```
<!DOCTYPE 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 / URL.

Including an External XML File in Another XML File: External Parsed Entities

Question

1. I am trying to include an external XML file in another XML file. Does Oracle Parser for Java v1 and v2 support external parsed entities?
2. We are using version 1.0, because that is what is shipped to the customers with release 10.7 and 11.0 of our application. Can you refer me to this, or some other sample code to do this.

   Shouldn't file b.xml be in the format:

   ```
   <?xml version="1.0" ?>
   <b>
     <ok/>
   </b>
   ```

   Does Oracle XML Parser come with a utility to parse an XML file and see the parsed output?

Answer

1. IE 5.0 will parse an XML file and show the parsed output. Just load the file like you would an HTML page.

   The following works, both browsing it in IE5 as well as parsing it with Oracle XML Parser v2. Even though I'm sure it works fine in Oracle XML Parser 1.0, you should be using the latest parser version as it is faster than v1.

   File: a.xml

   ```
   <?xml version="1.0" ?>
   <!DOCTYPE a [<!ENTITY b SYSTEM "b.xml">]>
    <a>&b;</a>
   ```

   File: b.xml

   ```
    <ok/>
   ```

When I browse / parse a.xml I get the following:

```
<a>
  <ok/>
</a>
```

2. Not strictly. The parsed external entity only needs to be a well-formed fragment. The following program (with xmlparser.jar from v 1.0) in your CLASSPATH shows parsing and printing the parsed document. It's parsing here from a String but the mechanism would be no different for parsing from a file, given it's URL.

```
import oracle.xml.parser.*;
import java.io.*;
import java.net.*;
import org.w3c.dom.*;
import org.xml.sax.*;
/*
** Simple Example of Parsing an XML File from a String
** and, if successful, printing the results.
**
** Usage: java ParseXMLFromString <hello><world/></hello>
*/
public class ParseXMLFromString {
  public static void main( String[] arg ) throws IOException, SAXException {
    String theStringToParse =
       "<?xml version='1.0'?>"+
       "<hello>"+
       "  <world/>"+
       "</hello>";
    XMLDocument theXMLDoc = parseString( theStringToParse );
    // Print the document out to standard out
    theXMLDoc.print(System.out);
  }
  public static XMLDocument parseString( String xmlString ) throws
   IOException, SAXException {
   XMLDocument theXMLDoc     = null;
    // Create an oracle.xml.parser.v2.DOMParser to parse the document.
    XMLParser theParser = new XMLParser();
    // Open an input stream on the string
    ByteArrayInputStream theStream =
         new ByteArrayInputStream( xmlString.getBytes() );
    // Set the parser to work in non-Validating mode
    theParser.setValidationMode(false);
```

```
    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;
  }
}
```

OraXSL Parser

Question

From where I can download oracle.xml.parser.v2.OraXSL?

Answer

It's part of our integrated XML Parser for Java V2 release. Our XML Parser, DOM, XPath implementation, and XSLT engine are nicely integrated into a single, cooperating package. http://technet.oracle.com/tech/xml/parser_java2/

XSL-T Processor and XSL Stylesheets

HTML Error in XSL

Question

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 you must know that every opening/starting tag should have an ending tag. So even the input that you are giving should have a matching ending tag, so you should modify your script like this:

```
<FORM>
<input type="text" name="country" size="15"> </input>
</FORM>
```

OR

```
<FORM>
<input type="text" name="country" size="15"/>
```

```
</FORM>
```

And also always remember, in XML the tags are case sensitive, unlike in HTML. So be careful.

Is <xsl:output method="html"/> Supported?

Question

Is the output method "html" supported inthe recent version of the XML/XSL parser?I was trying to use the <BR> tag with the<xsl utput method="xml"/> declaration butI got an XSLException error message indicating a not well-formed XML document. Then I tried the following output methoddeclaration: <xsl utput 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 utput method="html"/>
<xsl:template match="/">      <HTML>         <HEAD></HEAD>         <BODY>
<P>           Blah blah<BR>           More blah blah<BR>         </P>
</BODY>      </HTML>   </xsl:template>
```

My question is: "How do I use a not well-formed tags (like <IMG>, <BR>, etc.)in a XSL stylesheet?"

Answer

We fully support all options of <xsl utput> 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.<xsl utput method="html"/> requests that when the XSLT Engine *writes out* the result of your transformation, is a proper HTML document. What the XSLT engine reads *in* must be well-formed XML.

Question 2

Sorry for jumping in on this thread, but I have a question regarding your reply. I have an XSL stylesheet that preforms XML to HTML conversion. Everything works correctly with the exception of those HTML tags that are not well formed. Using your example if I have something like:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>
```

```
......
<input type="text" name="{NAME}" size="{DISPLAY_LENGTH}" maxlength="{LENGTH}">
</input>
......
</xsl:stylesheet>
```

It would render HTML in the format of

```
<HTML>......<input type="text" name="in1" size="10" maxlength="20"/>
......
</HTML>
```

While IE can handle this Netscape can not. Is there anyway to generate completely cross browser complient HTML with XSL?

Answer 2

If you are seeing:

```
<input ... />
```

instead of:

```
<input>
```

Then you are likely using the incorrect way of calling XSLProcessor.processXSL() since it appear that it's not doing the HTML output for you. Use:

```
void processXSL(style,sourceDoc,PrintWriter)
```

instead of:

```
DocumentFragment processXSL(style,sourceDoc)
```

and it will work correctly.

Netscape 4.0: Preventing XSL From Outputing <meta> Tag

Question

I'm using <xsl utput method="html" encoding="iso-8859-1" indent = "no" />. Is it possible to prevent XSL-T from outputting <META http-equiv="Content-Type" content="text/html; charset=iso-8859-1"> in the head because Netscape 4.0 has difficulties with this statement. It renders the page twice.

Answer

The XSLT 1.0 Recommendation says in Section 16.2 ("HTML Output Method")...If there is a HEAD element, then the html output method should add a META element immediately after the start-tag of the HEAD element specifying the character encoding actually used.

For example:

<HEAD><META http-equiv="Content-Type" content="text/html; charset=EUC-JP">.

So any XSLT 1.0-compliant engine needs to add this.

Question 2

Netscape 4.0 has following bug:

When Mozilla hits the meta-encoding tag it stops rendering the page and does a refresh. So you experience this anoying flickering. So I probably have to do a replacement in the servlets Outputstream, butI don't like doing so. Are there any alternatives.

Answer 2

Only alternatives I can think of are:

- Don't have a <HEAD> section in your HTML page. As per the XSL-T specification, this will suppress the inclusion of the <META> tag.
- Don't use method="HTML" for the output. Since it defaults to "HTML" as per the specification for result trees that start with <HTML> (in any mixture of case), you'd have to explicitly set it to method="xml" or method="text".

Neither is pretty, but either one might provide a workaround.

XSL Error Messages

Question

Where can I find more info on the XSL error messages. I get the error XSL-1900 , exception occurred. What does this mean ? How can I find out what caused the exception ?

Answer

If you are using Java, you could write Exception routines to trap errors.Using tools such as JDeveloper also helps.

The error messages of our components are usually more legible. XSL-1900 indicates possible internal error or incorrect usage. Please post a code sample that produces this error including your sample XML and DTD file if applicable.

Generating HTML: "<" Character

Question

I am trying to generate an HTML form for inputting data using column names from the user_tab_columns table and the following XSL code:

```
<xsl:template match="ROW">
<xsl:value-of select="COLUMN_NAME"/>
<: lt;INPUT NAME="<xsl:value-of select="COLUMN_NAME"/>>
</xsl:template>
```

although 'gt;' is generated as '>' 'lt;' is generated as '#60;'. How do I generate the "<" character?

Comment

Using the following:

```
<xsl:text disable-output-escaping="yes">entity-reference</xsl:text>
```

does what I need.

HTML "<" Conversion Works in oraxsl but not XSLSample.java?

Question

I can't seem to display HTML from XML.In my XML file I store the HTML snippet in an XML tag:

```
<PRE>
<body.htmlcontent>
<&#60;table width="540" border="0" cellpadding="0"
cellspacing="0">&#60;tr>&#60;td>&#60;font face="Helvetica, Arial"
size="2">&#60;!-- STILL IMAGE GOES HERE -->&#60;img
src="graphics/imagegoeshere.jpg"  width="200" height="175" align="right"
```

```
vspace="0" hspace="7">&#60;!-- END STILL IMAGE TAG -->&#60;!-- CITY OR TOWN NAME
GOES FIRST FOLLOWED BY TWO LETTER STATE ABBREVIATION -->&#60;b>City, state
abbreviation&#60;/b> - &#60;!-- CITY OR TOWN NAME ENDS HERE -->&#60;!-- STORY
TEXT STARTS HERE -->Story text goes here.. &#60;!-- STORY TEXT ENDS HERE
-->&#60;/font>&#60;/td>&#60;/tr>&#60;/table>
</body.htmlcontent>
</PRE>
```

I use the following in my XSL:

```
<xsl:value-of select="body.HTMLcontent" disable-output-escaping="yes"/>
```

However, the HTML output

```
<PRE>&#60;</PRE>
```

is still outputted and all of the HTML tags are displayed in the browser. How do I display the HTML properly?

Comment

That doesn't look right. All of the < are #60; in the code with an ampersand in front of them. They are still that way when they are displayed in the browser.

Even more confusing is that it works withoraxsl, but not with XSLSample.java.

Answer

This makes sense. Here's why:

- oraxsl internally uses the: void XSLProcessor.processXSL (style,source,printwriter);
- XSLSample.java uses:DocumentFragment XSLProcessor.processXSL (style,source);

The former supports <xsl:output> and all options related to writing out output that might not be valid XML (including the disable output escaping). The latter is pure XML-to-XML tree returned, so no <xsl:output> or disabled escaping can be used since nothing's being output, just a DOM tree fragment of the result is being returned.

XSL-T Examples

Question

Is there any site which has good examplesor small tutorials on XSL-T?

Answer

This site is an evolving tutorialon lots of different XML / XSLT / XPath-related subjects:

http: / / zvon.vscht.cz / ZvonHTML / Zvon / zvonTutorials_en.html

XSL-T Features

Question

1. Is there a list of features of the XSL-T that Oracle XDK implements?
2. So the v2 parsers implement more features of the recommendation than IE5 ? My first impression supports this, the use of <xsl:choose... and <xsl:if... works with the v2 parser but gives strange messages with IE5.

Answer

1. Our v2 parsers support the W3C Recommendation of w3c XSL-T version 1.0 at http: / / www.w3.org / TR / XSLT.
2. You are correct. Ours is XSL-T Recommendation compliant.

Using XSL To Convert XML Document To Another Form

Question

I am in the process of trying to convert an xml document from one format to another by means of an xsl (or xslt) stylesheet. Before incorporating it into my java code, I tried testing the transformation from the command line:

```
> java oracle.xml.parser.v2.oraxsl jwnemp.xml jwnemp.xsl newjwnemp.xml
```

The problem is that instead of returning the transformed xml file (newjwnemp.xml), the above command just returns a file with the xsl code from jwnemp.xsl in it. I cannot figure out why this is occurring. I have attached the two input files.

```
<?xml version="1.0"?>
```

```
<employee_data>
   <employee_row>
      <employee_number>7950</employee_number>
      <employee_name>CLINTON</employee_name>
      <employee_title>PRESIDENT</employee_title>
      <manager>1111</manager>
      <date_of_hire>20-JAN-93</date_of_hire>
      <salary>125000</salary>
      <commission>1000</commission>
      <department_number>10</department_number>
   </employee_row>
</employee_data>

<?xml version='1.0'?>
<ROWSET xmlns:xsl="HTTP://www.w3.org/1999/XSL/Transform">
   <xsl:for-each select="employee_data/employee_row">
   <ROW>
      <EMPNO><xsl:value-of select="employee_number"/></EMPNO>
      <ENAME><xsl:value-of select="employee_name"/></ENAME>
      <JOB><xsl:value-of select="employee_title"/></JOB>
      <MGR><xsl:value-of select="manager"/></MGR>
      <HIREDATE><xsl:value-of select="date_of_hire"/></HIREDATE>
      <SAL><xsl:value-of select="salary"/></SAL>
      <COMM><xsl:value-of select="commission"/></COMM>
      <DEPTNO><xsl:value-of select="department_number"/></DEPTNO>
   </ROW>
   </xsl:for-each>
</ROWSET>
```

Answer

This is occurring nearly 100%-likely because you have the wrong XSL namespace uri for your xmlns:xsl="..." namespace declaration.

If you use: xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

everything works.

If you use xmlns:xsl="-- any other string here --"

If will do what you're seeing.

Information on XSL?

Question

I cannot find anything about using XSL. Can you help? I would like to get an XML and XSL file to show my company what thay can expect from this technology. XML alone is not very impressive for users.

Answer

A pretty good starting place for XSL is the following page:

http://metalab.unc.edu/xml/books/bible/updates/14.html

It shows pretty much in english what the jist of xsl is. XSL isn't really anything more than an XML file anyways, so I don't think that it will be anymore impressive to show to a customer. There's also the main website for xsl which is:

http://www.w3.org/style/XSL/

XSLProcessor and Multiple Outputs?

Question

I recall seeing discussions about XSLProcessor producing more than one result from one XML and XSL. How can this can be achieved?

Answer

XML Parser 2.0.2.8 supports <ora:output> to handle this.

Good Books for XML/XSL

Question

Can any one suggest good books for learning about XML/XSL?

Answer

There are many excellent articles, whitepapers, and books that describe all facets of XML technology. Many of these are available on the world wide web. The following are some of the most useful resources we have found:

- XML, Java, and the Future of the Web by Jon Bosak, Sun Microsystems http://metalab.unc.edu/pub/sun-info/standards/xml/why/xmlapps.htm
- XML for the Absolute Beginner by Mark Johnson, JavaWorld http://www.javaworld.com/jw-04-1999/jw-04-xml_p.html
- XML And Databases by Ronald Bourret, Technical University of Darmstadt http://www.informatik.tu-darmstadt.de/DVS1/staff/bourret/xml/
- XMLAndDatabases.htm World Wide Web Consortium (W3C)
- XML Specifications http://www.w3.org/XML/
- XML.com (a broad collection of XML resources and commentary) http://www.xml.com/
- Annotated XML Specification by Tim Bray, XML.com http://www.xml.com/axml/testaxml.htm
- The XML FAQ by the W3C XML Special Interest Group http://www.ucc.ie/xml/ XML.org (the industry clearing house for XML DTDs that allow companies to exchange XML data)
- http://xml.org/
- xDev (the DataChannel XML Developer pages) http://xdev.datachannel.com/

Version Number of XDK?

Question

How do I determine the version number of the XDK toolkit that I downloaded?

Answer

You can find out the full version number by looking at the readme.html file included in the archive and linked off of the Release Notes page.

Including Binary Data in an XML Document

Question

How do I include binary data in an XML document?

Answer

There is no way to directly include binary data within the document; however, there are two ways to work around this:

- The binary data could be referenced as an external unparsed entity that resided in a different file.
- The binary data can be uuencoded (meaning converting binary data into ASCII data) and be included in a CDATA section. The limitation on the encoding technique is to be sure that it only produces legal characters for the CDATA section.

Converting XML to HTML

Question

How do I convert XML files into HTML files?

Answer

You need to create an XSL stylesheet to render your XML into HTML. You can start with an HTML document in your desired format and populated with dummy data. Then you can replace this data with the XSLT commands that will populate the HTML with data from the XML document completing your stylesheet.

XML Developer Kits for HP/UX Platform

Question

I would like to know if there are any release plans for the XML Parser or an XDK for HP/UX platform.

Answer

HP-UX ports for our C/C++ Parser as well as our C++ Class Generator are available. Look for an announcement on http://technet.oracle.com

18

Using XML Java Class Generator

This chapter contains the following sections:

Accessing XML Java Class Generator

XML Java Parser Class Generator is provided with Oracle8*i* and is also available for download from the OTN site: http: / / technet.oracle.com / tech / xml.

It is located at $ORACLE_HOME / xdk / java / classgen

Using XML Class Generator for Java

The XML Class Generator for Java creates Java source files from an XML DTD. This is useful for the following situations:

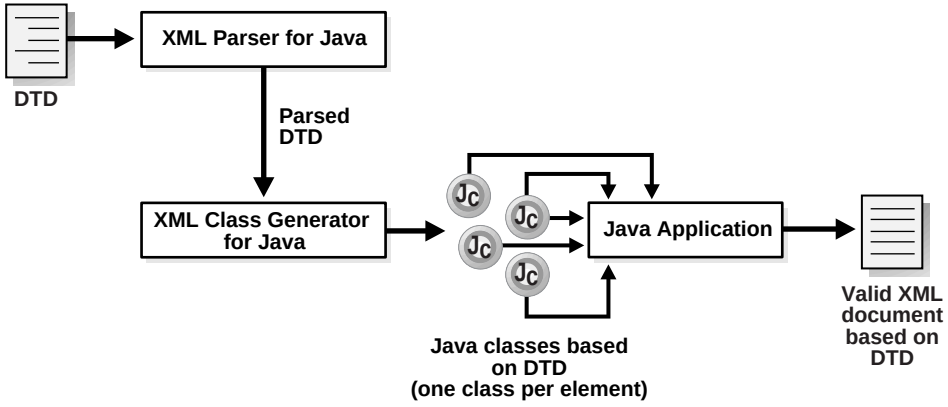
- When an application wants to send an XML message to another application based on an agreed-upon DTD
- As the back end of a web form to construct an XML document.

Using the Class Generator classes, Java applications can construct, validate, and print XML documents that comply with the input DTD.

It also optionally generates Javadoc comments in the source files. XML Class Generator requires Oracle XML Parser for Java.

Figure 18–1 shows a summary of how XML Java Class Generator is used.

Figure 18–1 XML Java Class Generator



XML Java Class Generator works in conjunction with XML Parser for Java, which parses the DTD and passes the parsed document to the Class Generator.

See Also: Figure 3–8, "Using Oracle XML Components to G enerate an XML Document - C Options" in Chapter 3, "Oracle XML Components and General FAQs".

Figure 18–2 shows the XML Java Class Generator usage.

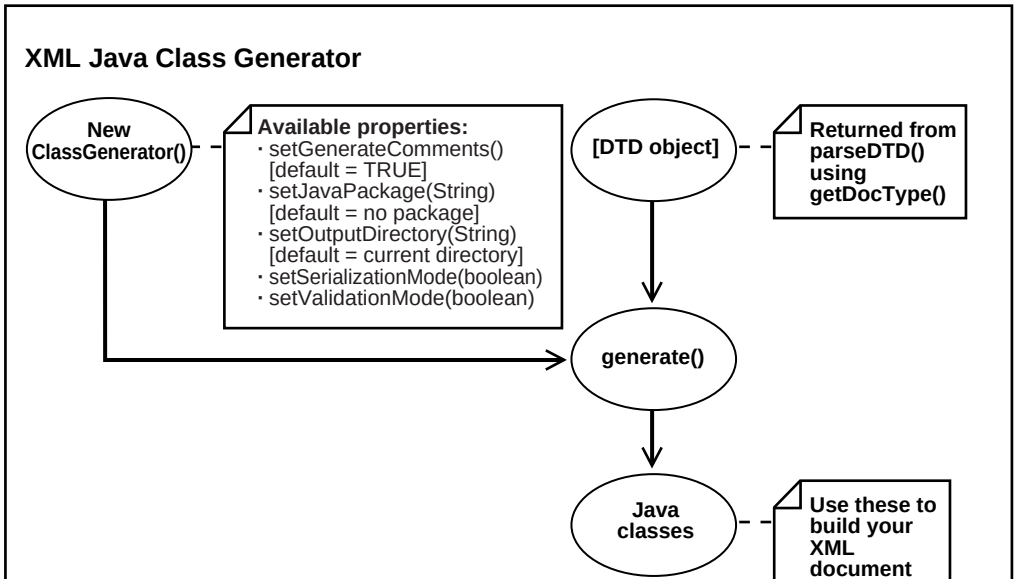
XML Java Class Generator operates as follows:

1. A new ClassGenerator() class is initiated and inputs the generate() method. Available properties for ClassGenerator() class 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 17–4, "XML Parser for Java: DOMParser()" on page 17-14 from Chapter 17, "Using XML Parser for Java".
3. The generate() method generates Java classes which can then be used to build your XML document.

See Also:

- Appendix C, "XDK for Java: Specifications and Cheat Sheets"
- Oracle8i XML Reference
- Chapter 3, "Oracle XML Components and General FAQs", "Using Oracle XML Components to Generate XML Documents: Java" on page 3-17

Figure 18–2 XML Java Class Generator Usage



XML Java Class Generator Examples

The following three examples illustrate:

- How the XML Class Generator for Java is used to process a DTD and generate classes for the DTD's elements.
- How element class methods are used to construct a valid XML document

The following examples are included here:

Example Input DTD

The DTD file for employee data, `employee.dtd`, is used as the input to the class generator. Here, the DTD specifies that the XML document root is EMP and the row element is EMP_ROW. EMP consists of one or more EMP_ROWs.

Each EMP_ROW contains a required EMPNO attribute for the employee number, as well as several optional attributes such as ENAME for employee names, JOB for job type, MGR for manager, and so on. Optional attributes are followed by a "?" in the element definition.

XML Java Class Generator DTD Example 1: Employee Data

```
<!-- DTD for Employee Data -->
<!ELEMENT EMP (EMP_ROW)*>
<!ELEMENT EMP_ROW (EMPNO, ENAME?, JOB?, MGR?, HIREDATE?, SAL?,
                   COMM?, DEPTNO?)>
<!ATTLIST EMP_ROW ROWNO CDATA #REQUIRED>
<!ELEMENT EMPNO (#PCDATA)>
<!ELEMENT ENAME (#PCDATA)>
<!ELEMENT JOB (#PCDATA)>
<!ELEMENT MGR (#PCDATA)>
<!ELEMENT HIREDATE (#PCDATA)>
<!ELEMENT SAL (#PCDATA)>
```

```
<!ELEMENT COMM (#PCDATA)>
<!ELEMENT DEPTNO (#PCDATA)>
```

XML Java Class Generator Example 1: Processing the DTD to Generate Java Classes

The following example processes the DTD and generates the corresponding classes for elements in the DTD. Running the class generator on the DTD described in "XML Java Class Generator DTD Example 1: Employee Data" on page 18-5, creates Java classes for each element, EMP, EMP_ROW, EMPNO, ENAME, and so on.

A Java application can then use the methods defined on these classes to create a valid XML document containing employee data.

```
import java.io.*;
import java.net.*;
import oracle.xml.parser.*;
import oracle.xml.classgen.*;
import org.w3c.dom.Element;

public class SampleMain
{
  public SampleMain()
  {
  }
  public static void main (String args[])
  {
     // validate 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 ;
    }
    try // to open the External DTD File
    {
      // instantiate the parser
      XMLParser parser = new XMLParser();

      if (args.length == 3)
        parser.parseDTD(fileToURL(args[2]), args[1]);
      else
```

```
      parser.parse(fileToURL(args[0]));

    XMLDocument doc = parser.getDocument();
    DTD 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...
    ClassGenerator generator = new ClassGenerator();

    // 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)
    {
      // Can only happen if the file
      // protocol were not recognized
      throw new Error("unexpected MalformedURLException");
    }
  }
}
```

XML Java Class Generator Example 2: Creating a Valid XML Document from Java Classes

This Java example shows how generated methods can be used. Two row elements are created, emp_row1 and emp_row2. Elements for each column are also created, empno1, ename1, and so on.

To build an XML document tree, the various data elements are grouped by assigning them to each row element as tree nodes. Each row element is then added as a node to the document root element EMPLIST.

In this example, classes generated by the Class Generator, are in uppercase:

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

public class CreateEmployees
{
  public static void main (String args[])
    {
      try
      {
        EMP EMPLIST = new EMP();
        DTD dtd = EMPLIST.getDTDNode(); // get static from base document

        // New employee emp_row1
        EMP_ROW emp_row1 = new EMP_ROW(1); // create row and set ROWNO
        EMPNO empno1 = new EMPNO("7654");
        ENAME ename1 = new ENAME("MARTIN");
        JOB job1 = new JOB("SALESMAN");
```

```
            MGR mgr1 = new MGR("7698");
            HIREDATE hiredate1 = new HIREDATE("1981-09-28 00:00:00.0");
            SAL sal1= new SAL("1250");
            COMM comm1= new COMM("1400");
            DEPTNO deptno1 = new DEPTNO("30");

            // New employee emp_row2
            EMP_ROW emp_row2 = new EMP_ROW(2); // create row and set ROWNO
            EMPNO empno2 = new EMPNO("7788");
            ENAME ename2 = new ENAME("SCOTT ");
            JOB job2 = new JOB("ANALYST ");
            MGR mgr1 = new MGR("7566");
            HIREDATE hiredate2 = new HIREDATE("1987-04-19 00:00:00.0");
            SAL sal2= new SAL("3000");
            COMM comm2= new COMM("");
            DEPTNO deptno2 = new DEPTNO("20");

            emp_row1.addnode(empno1); // Add data as tree nodes to emp_row1
            emp_row1.addnode(ename1);
            ...

            emp_row2.addnode(empno2); // Add data as tree nodes to emp_row2
            emp_row2.addnode(ename2);
            ...

            EMPLIST.addNode(emp1); // Add emp_row1 as tree node to
                                   // EMPLIST doc root
            EMPLIST.addNode(emp2); // Add emp_row2 as tree node to
                                   // EMPLIST doc root

            EMPLIST.validateContent();
            EMPLIST.print(System.out);
          }
        catch (Exception e)
        {
            System.out.println(e.toString());
            e.printStackTrace();
        }
      }
    }
```

XML Java Class Generator Example 3: Resulting XML Document Built by a Java Application

The previous Java application, "XML Java Class Generator Example 2: Creating a Valid XML Document from Java Classes", creates an XML document similar to the following:

```
<?xml version="1.0"?>
<!DOCTYPE EMP SYSTEM "employee.dtd">
<EMP>
  <EMP_ROW ROWNO = "1">
    <EMPNO>7654</EMPNO>
    <ENAME>MARTIN</ENAME>
    <JOB>SALESMAN</JOB>
    <MGR>7698</MGR>
    <HIREDATE>1981-09-28 00:00:00.0</HIREDATE>
    <SAL>1250</SAL>
    <COMM>1400</COMM>
    <DEPTNO>30</DEPTNO>
  /EMP_ROW>
  <EMP_ROW ROWNO = "2">
    <EMPNO>7788</EMPNO>
    <ENAME>SCOTT</ENAME>
    <JOB>ANALYST</JOB>
    <MGR>7566</MGR>
    <HIREDATE>1987-04-19 00:00:00.0</HIREDATE>
    <SAL>3000</SAL>
    <COMM></COMM>
    <DEPTNO>20</DEPTNO>
  </EMP_ROW>
</EMP>
```

XML Java Class Generator Sample Files in sample/

Table 18–1 lists the sample files available in directory of $ORACLE_HOME/xdk/java/classgen/sample that you receive with the Oracle software.

Table 18–1 XML Java Class Generator: Sample Files in sample/

Sample File Name	Description
SampleMain.java	A sample application to generate Java source files based on a DTD
Widl.dtd	A sample DTD
Widl.xml	A sample XML file based on Widl.dtd
TestWidl.java	A sample application to construct XML document using the Java source files generated by SampleMain
Widl.out	XML Document printed by TestWidl.out

How to Run the XML Java Class Generator Samples in sample/

To run the sample program, use the following command from the sample/ directory:

```
make target 'demo'
```

Then follow these three steps to run the sample code:

1. Compile and run `SampleMain` to generate the Java source files.

   Set the CLASSPATH to contain 'classgen.jar', 'xmlparserv2.jar', and current directory.

   - javac SampleMain.java
   - java SampleMain -root WIDL Widl.dtd

   or

   - java SampleMain Widl.xml

2. Compile the Java source files generated by SampleMain:
   - BINDING.java
   - CONDITION.java
   - REGION.java

- SERVICE.java
- VARIABLE.java
- WIDL.java.

3. Run the test application to print the XML Document.
   - javac TestWidl.java
   - java TestWidl

XML Java CLass Generator, DTD Example 1: DTD Input — widl.dtd

The following example, `widl.ddt` is the DTD file referenced by the XML file `widl.xml`.

```
<!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 Java CLass Generator, XML Example 1: XML Input — widl.xml

This XML file inputs SampleMain.java:

```
<?xml version="1.0"?>
<!DOCTYPE WIDL SYSTEM "Widl.dtd">
<WIDL>
    <SERVICE NAME="sname" URL="surl"/>
    <BINDING NAME="bname"/>
</WIDL>
```

XML Java CLass Generator, Java Example 1: SampleMain.java

`SampleMain.java` generates a Java source file based on the input DTD, `widl.ddt` or `widl.xml`.

```
import java.io.*;
import java.net.*;
import oracle.xml.parser.v2.*;
import oracle.xml.classgen.*;
import org.w3c.dom.Element;

/**
 * Sample Class to uses XML Class Generator
 */
```

```
public class SampleMain
{

   public SampleMain()
   {
   }

   public static void main (String args[])
   {
       // validate 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 ;
      }
      try // to open the XML File/ External DTD File
      {
         // 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...
      ClassGenerator generator = new ClassGenerator();

      // 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 Java CLass Generator, Java Example 2: TestWidl.java

TestWidl.java constructs an XML document using the Java source files generated by SampleMain.java.

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

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 Java CLass Generator, XML Example 2: DTD Input — 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>
```

Frequently Asked Questions (FAQs) : Class Generator for Java

This section lists XML Java Class Generator questions and answers.

Automatic Population

Question

I used XML Java Class Generator to generate some class files for a sample DTD schema. On looking at the classes generated, I saw only APIs to construct instances through regular means (providing values to members) and APIs to print the XML representation to an outputstream. The functionality is not as complete as I expected it to be. I expected the document class to include a constructor that will take an XML document object and populate the various members automatically.

Does this functionality exist?

Answer

XML Java Class Generator does not have this functionality at this time as only the XML Parser for Java can build DOM trees.

XML to Java Object Mapping

Question

I want to use XML as the storage medium for meta-data. This meta-data consists of complex, nested structures for which XML is well suited. At system start-up time I need to have access to all these meta-data objects as first class Java objects.However, I don't want to have to write custom code to run atop an XML parser such as Oracle XML Parser to build instances of these objects (one XML file -> one instance).

My ideal interface looks like this:

DTD / XML Schema + XML files fed to an XMLClassGenerator (runs atop an XMLParser) spits out Java source code that instantiates my meta-data object.

For example, if I have a meta.dtd (schema) and meta.xml which I want to map to a Meta.java.class Meta {public String metaName;public int numMeta;public Date dateMeta;};

I want to be able to run XMLClassGenerator on meta.dtd (schema) and meta.xml and get this kind of source code output:

```
MetaBuilder.javaclass MetaBuilder {public init()
 {
    Meta metaInstance = new Meta();
    metaInstance.metaName = "Example1";
    metaInstance.numMeta = 1;
    metaInstance.dateMeta = new Date("07/16/1999");}
```

Do you plan to extend the Class Generator capability to do this? I can use this to send Java objects across a network, instead of an XML document over RMI.

Answer

I also am not aware of a direct solution out there. This would be an interesting extension for our Class Generator.

DTD Class Generator: Which Child Classes Work

Question

In transcribing the code for the DTD Class Generator code form an article, I encountered an error I do not know how to correct. The program attempts to instantiate an XMLParser object, whose class is defined as abstract. What child classes of this class would work for running the code?

Answer

XMLParser was concrete in our V1 XML Parser. It's now abstract in V2 and you must either construct a DOMParser or a SAXParser. The article is probably using a DOMParser.

Installing XML Class Generator

Question

What steps install XML Java Class Generator?

Answer

There are basically two steps:

1. Download the XML Class Generator for Java
2. Set CLASSPATH to include both classgen.jar from the /bin directory and xmlparserv2.jar from the XML Parser for Java v2 download.

Role of the XML Class Generator for Java

Question 1

What does the XML Class Generator for Java do?

Answer 1

The XML Class Generator for Java creates Java source files from an XML DTD. This is useful when an application wants to send an XML message to another application based on an agreed-upon DTD or as the back end of a web form to construct and XML document. Using these classes, Java applications can construct, validate, and print XML documents that comply with the input DTD. The Class Generator works in conjunction with the Oracle XML Parser for Java v2, which parses the DTD and passes the parsed document to the class generator.

Question 2

How do I use the XML Class Generator for Java to get XML data?

Answer 2

First, get the data from the database using JDBC ResultSets. Then, instantiate objects using the classes generated by the XML Class Generator.

Automatic Instantiation of Objects Based on XML File: Using the XML Class Generator

Question

Can I automatically instantiate objects based on an XML file?

Answer

Presently, you can only use DTD to generate classes and then instantiate each object by calling the create methods in the class. Please contact us if you have a business need to use XML file to instantiate objects automatically.

Which DTD's are Supported?

Question

Does XML Java Class Generator support any kind of DTD?

Answer

Yes, it supports any kind of DTD that is XML1.0 compliant.

Using the XML Class Generator Samples

Question

How do I fix the classes-not-found errors?

Answer

To fix the classes-not-found error, include classgen.jar and xmlparserv2.jar in your CLASSPATH.

Class Generator: Cannot Create Root Object More than Once

Question

I've generated, from a DTD, a set of Java classes with "ClassGenerator". After, I've tried to create a Java application that uses these classes to create an XML file from data passed as arguments. I cannot create the root object, the object derived from CGDocument, more than one time because I obtain the following error message :

```
oracle.xml.parser.XMLDOMException: Node doesn't belong to the current document
```

How do I handle the star operator (*). When the application starts I do not know how many times the element will appear. Thus I do not build a static loop where I make a sequence of "element.addNode()". The problem is that some of these will be empty and I will obtain an XML document with a set of empty elements with empty attributes.

Answer

You can create subsequent XML docs by calling the constructor each time. A well-formed XML document is not permitted to have more than one root node,

therefore you cannot use the "*" on the element you are designating as the document root.

Class Generator:Creating XML Files from Scratch, Using the DOM API

Question

I want to create an XML file using the DOM API. I dont 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. I do not know how to start! 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.

19

Using XSQL Servlet

This chapter contains the following sections:

- Using XSQLConfig.xml to Tune Your Environment
- Limitations
- Frequently Asked Questions (FAQs) - XSQL Servlet

Accessing XSQL Servlet

XSQL Servlet is provided with Oracle8*i* and *is* also available for download from the OTN site: http://technet.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 Servlet

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://technet.oracle.com/tech/xml/xsql_servlet/listing.htm

XSQL Servlet Features

XSQL Servlet uses XML Parser for JavaV2 and XML- SQL Utility to generate XML pages for SQL queries. It can combine the power of SQL, XML, and XSLT in the server with HTTP protocol as a transport mechanism, to do the following tasks:

- Receive web-based information requests from any client device on the Web
- Query a logical "view" of business data requested anywhere
- Return the "datagram" in XML over the web to the requester, or optionally
- Transform the information into any XML, HTML, or text format required

XSQL Pages bring this capability to you by automating the use of underlying XML components to solve common cases with no programming involved.

> **See Also:** Appendix B, "XDK for Java: Specifications and Cheat Sheets" for the XSQL Servlet specifications and cheat sheets, and the XSQL Servlet Release Notes on OTN at http://technet.oracle.com/tech/xml

Introducing Oracle XSQL Pages

Oracle XSQL Pages are templates that allow anyone familiar with SQL to declaratively:

- Assemble dynamic XML "data pages" based on one or more parameterized SQL query

- Transform the "data page" to produce a final result in any desired XML, HTML, or Text-based format using an associated XSLT Transformation

For example, to serve a real-time XML "datagram" from Oracle8*i* of all available flights landing today at JFK airport just build an XSQL Page named `AvailableFlightsToday.xsql` that looks like the example below. Use your favorite text editor…

```
<?xml version="1.0"?>
<xsql:query connection="demo" 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 = '{@City}'
  ORDER BY ExpectedTime
</xsql:query>
```

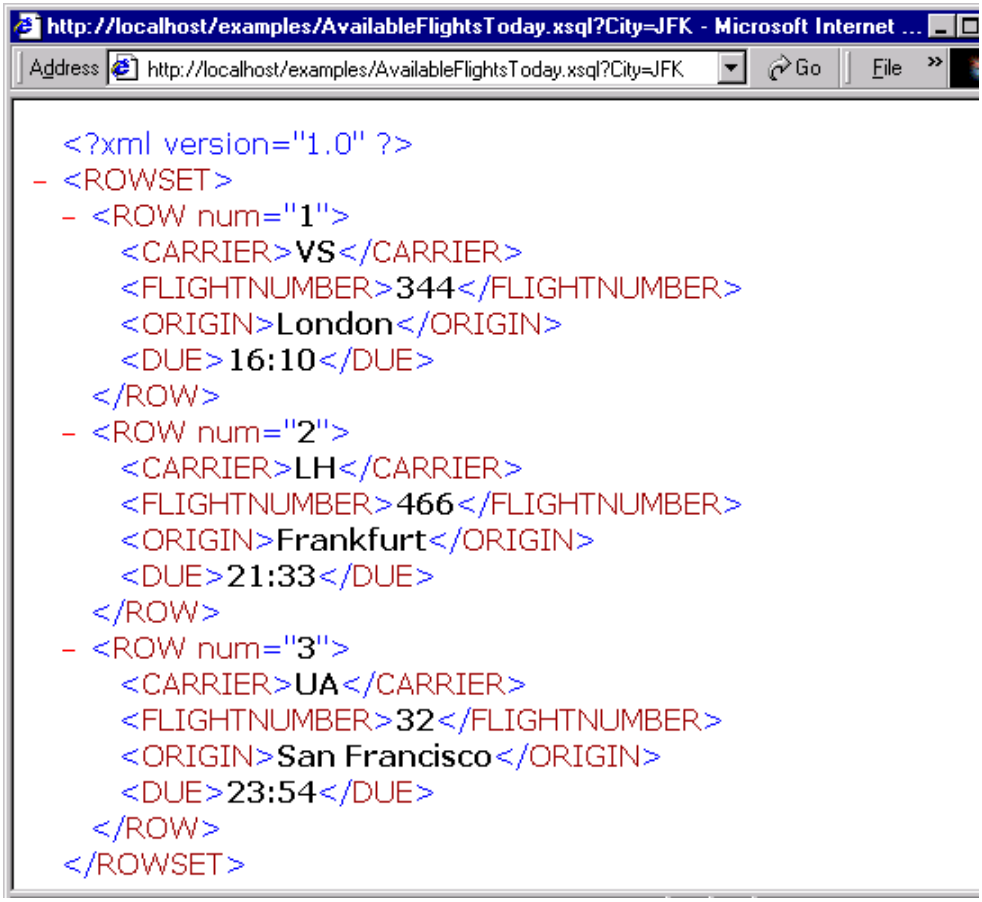
Save the file, and then browse the URL:

```
http://yourcompany.com/AvailableFlightsToday.xsql?City=JFK
```

By installing the XSQL Servlet on your favorite web server, and copying the `AvailableFlightsToday.xsql` file to that server's virtual root directory, you are already in business.

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 19–1.

Figure 19–1 XML Result From XSQL Page (AvailableFlightsToday.xsq) Query



If the canonical <ROWSET> and <ROW> XML output from Figure 19–1 is not the XML format that the requestor wants, then use an XSLT stylesheet to transform this XML document into the XML document format desired.

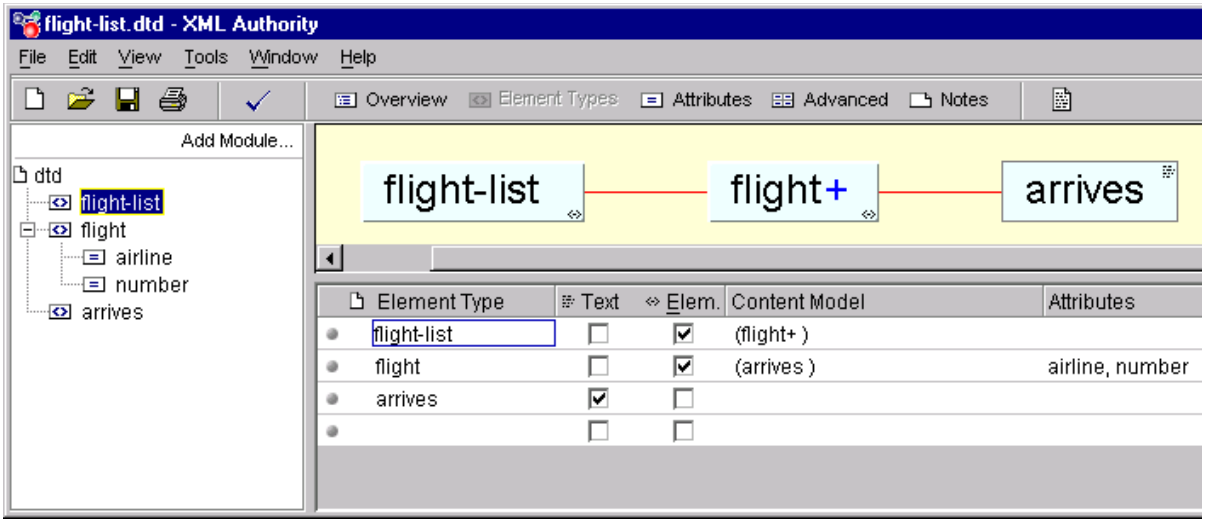
Typically the desired format is communicated to you to by a Document Type Descriptor (DTD). A DTD is in effect, a "schema" definition. It formally defines what XML content the documents of that type can have.

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 2.

Exploring the "industry-standard" flight-list.dtd using Extensibility's XML Authority 1.1

Figure 19–2 Exploring the 'industry standard' flight-list.dtd using Extensibility's XML Authority



This shows that the standard XML formats for Flight Lists are:

- `<flight-list>` element, containing one or more…
- `<flight>` elements, having attributes airline and number, each of which contains an…
- `<arrives>` element.

By associating the following XSLT stylesheet, flight-list.xsl, with the XSQL Page, you can "morph" the default <ROWSET> and <ROW> format of your arriving flights query results 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 "action elements" 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.

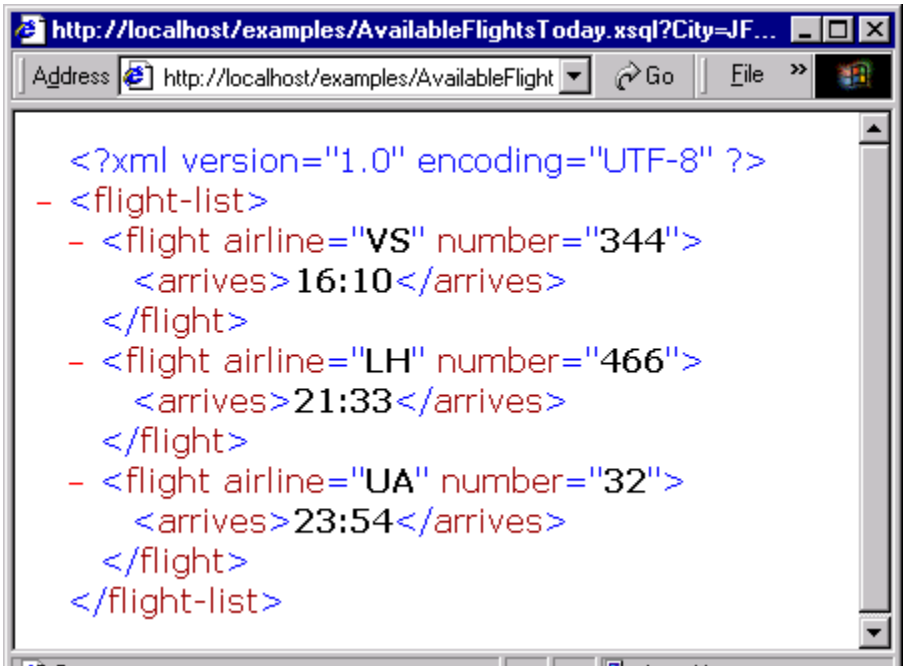
Associate the stylesheet to your XSQL Page by adding an `<?xml-stylesheet?>` processing instruction to the top of the page as follows:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="flight-list.xsl"?>
<xsql:query connection="demo" 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 = '{@City}'
  ORDER BY ExpectedTime
</xsql:query>
```

This 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 19–3.

Figure 19–3 XSQL Page Results in "Industry Standard" XML Format

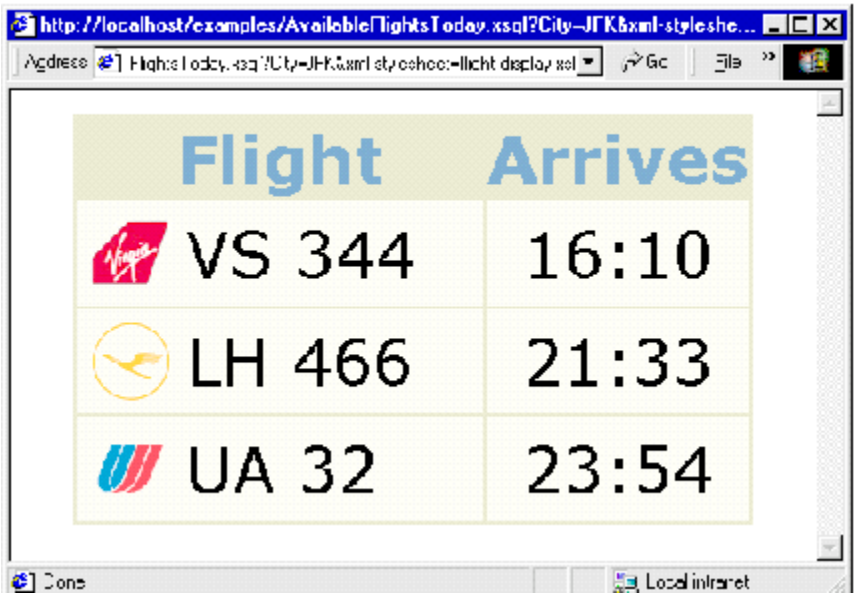


To return the same XML information in HTML instead, simply use a different XSLT stylesheet.

Rather than producing elements like <flight-list> and <flight>, your stylesheet produces HTML elements like <table>, <tr>, and <td>.

The result of the dynamically queried information then looks like the HTML page shown in Figure 19–4. 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 19–4 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>
    <style> td {font-family:verdana,arial; font-size:40;
            background-color:#f7f7e7; color:#000000 }
      th,table {font-family:verdana,arial; font-size:40;
            background-color:#cccc99; color:#336699 }
```

```
  </style>
  </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 (e.g. <td>…</td>) and that empty tags use the XML empty element syntax <br/> instead of just <br>.

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.

For more information on XSQL Pages refer to the XDK for Java, XSQL Servlet Release Notes that accompany the software download from the Oracle Technology Network at http://technet.oracle.com/tech/xml

Oracle XSQL Pages: Setup and Examples

Once you have built a set of XSQL Pages, you can use your pages by doing the following:

- Requesting them over the Web after installing the Oracle XSQL Servlet on your favorite Web Server,
- Calling the Oracle XSQL Command Line Utility in batch programs,
- Invoking the XSQLRequest.process() method from within any Java program
- Using `<jsp:include page="foo.xsql">` or `<jsp:forward page="bar.xsql">` in a JSP

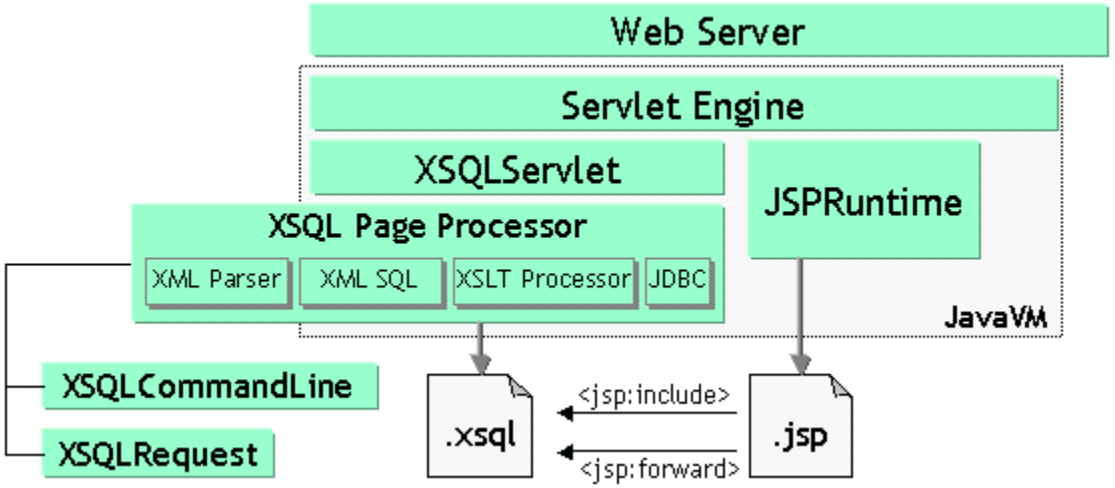
Figure 19–5 shows how XSQL Servlet is one of three "clients" to the core XSQL Page Processor Engine. This is also illustrated in Figure 19–6 and Figure 19–8.

XSQL Page Processor interprets, caches, and processes the contents of your XSQL Pages. As its name implies, XSQLServlet is a Java Servlet that allows XSQL Pages to be accessed through a web server over HTTP.

The other two clients are:

- XSQLCommandLine utility that allows XSQL Pages to be processed from the command line
- XSQLRequest class that offers a simple API to process XSQL Pages programmatically from within a custom Java program

Figure 19–5 Installing XSQL Servlet — Explaining XSQL Servlet Architecture



Setting the CLASSPATH Correctly

The XSQLServlet, XSQLRequest, and XSQLCommandLine, as well as the core XSQLPageProcessor are all Java programs and as such 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:

- classes111.zip for the Oracle JDBC Driver
- xmlparserv2.jar for the Oracle XML Parser for Java V2
- oraclexmlsql.jar for the Oracle XML-SQL Utility for Java
- oraclexsql.jar for the Oracle XSQL Servlet and Page Processor

In addition, the directory where XSQL Page Processor's configuration file (XSQLConfig.xml) resides must also be listed as a directory in the CLASSPATH. This usually resides in the ./xsql/lib directory.

So, putting all this together, if you have installed the XSQL distribution in C:\xsql, then your CLASSPATH would appear as follows:

- On Windows NT:

  ```
  C:\xsql\lib\classes111.zip;C:\xsql\lib\xmlparserv2.jar;
  C:\xsql\lib\oraclexmlsql.jar;C:\xsql\lib\oraclexsql.jar;C:\xsql\lib
  ```

- On UNIX, if you installed in your /web directory, the CLASSPATH would appear as follows:

  ```
  /web/xsql/lib/classes111.zip:/web/xsql/lib/xmlparserv2.jar:
  /web/xsql/lib/oraclexmlsql.jar:/web/xsql/lib/oraclexsql.jar:/web/xsql/lib
  ```

To use the XSQL Servlet, one additional setup step is required: associating 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.

For the purposes of this lesson, we'll be using the Oracle Web-to-Go single-user web server that is small, easy to use, and that supports Java Servlets.

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>
    </connection>
    <connection name="prod">
      <username>prodapp</username>
      <password>sumpti0us</password>
<dburl>jdbc:oracle:thin:@localhost:1521:prodDB</dburl>
      <driver>oracle.jdbc.driver.OracleDriver</driver>
    </connection>
</connectiondefs>
:
```

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").

Single Query Page

For a page with a single query this looks like:

```
<?xml version="1.0"?>
<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
    SELECT * FROM DUAL
</xsql:query>
```

Multiple Query Page

For a page with more than one query, it looks like the following:

```
<?xml version="1.0"?>
<page connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:query>SELECT 'X' as X FROM DUAL</xsql:query>
  <xsql:query>SELECT 'Y' as Y FROM DUAL</xsql:query>
```

```
</page>
```

You can use these techniques to publish an XML-based product catalog.

XSQL Page Common Tags

Table 19–1 lists three useful and common XSQL Servlet tags. For a full list of tags see *Oracle8i XML Reference*.

Table 19–1 XSQL Common Tags

XSQL Tag Examples	Description
<xsql:query>	To include XML-based SQL query result in the template wherever you want, use <xsql:query> tags.
<?xml-stylesheet?>	To associate an XSL stylesheet to the page, add <?xml-stylesheet?> instructions line at the top of the file.
<xsql:action>	To extend the set of actions that can be performed to assemble the "datapage", use the <xsql:action> element.

Save the file and request it through your browser to get immediate results.

Additional XSQL Page Example

Here is another simple example of an XSQL Page. This could be in response to a URL request such as:

```
http://yourcompany.com/AvailableFlightsToday.xsql?City=NYC
```

An example XSQL Page:

```
<?xml version="1.0"?>
  <xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
     SELECT Carrier, FlightNumber, Origin, TO_CHAR(ExpectedTime,'HH24:MI') Due
        FROM FlightSchedule
        WHERE TRUNC(ArrivalTime) = TRUNC(SYSDATE)
        AND Destination = '{@City}'
        ORDER BY ExpectedTime
  </xsql:query>
```

To return the information in HTML or other XML format that complies with a DTD, add a <?xml-stylesheet....?> tag such as:

```
<?xml version="1.0"?>
```

```
<?xml-stylesheet type="text/xsl" href="FlightList.xsl"?>
 <xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
    SELECT Carrier, FlightNumber, Origin, TO_CHAR(ExpectedTime,'HH24:MI') Due
      FROM FlightSchedule
      WHERE TRUNC(ArrivalTime) = TRUNC(SYSDATE)
      AND Destination = '{@City}'
      ORDER BY ExpectedTime
</xsql:query>
```

Note: {@City} is a parameter based on City=NYC. The parameter is made up of actions comprised of handles. The actions make it easy to assemble and transform information. The handles allow stylesheets to be transformed.

Built-in XSQL Action Elements

Table 19–2 lists several supplied built-in XSQL Action Elements

Table 19–2 Built-in XSQL Action Elements

Action Element	Description
<xsql:ref-cursor-function>	Includes the canonical XML representation of the result set of a cursor returned by a PL/SQL stored function. Use the new dynamic SQL features that are built-in to PL/SQL in Oracle8i, with <xsql:set-page-param>.
<xsql:set-page-param>	Set a page-level (local) parameter that can be referred to in subsequent SQL statements in the page. The value can be set using a static value, the value of another parameter, or the results of a SQL statement.
<xsql:include-param>	Include a parameter and its value as an element in your XSQL page.
<xsql:set-session-param>	Set an HTTP-Session level parameter. The value can be set using a static value, the value of another parameter, or the results of a SQL statement.
<xsql:set-cookie>	Set an HTTP Cookie. The value can be set using a static value, the value of another parameter, or the results of a SQL statement.
<xsql:insert-param>	Inserts the value of a single parameter containing XML. Can optionally supply a transform to get it in canonical format.

Using the XSQLCommandLine Utility

With the XSQLCommandLine Utility you can post XML to the XSQL page it is processing (typically, for inserting into the database) by specifying the filename or URL of the XML to be posted as the value of the posted-xml command line parameter. For example, posted-xml=filename or posted-xml=http://somesite/someurl are both valid.

Custom action handlers can now refer to the content of the posted XML document any number of times necessary when calling getPageRequest().getPostedDocument(). Previously, the second time calling this returned null.

The XSQLRequest class, the programmatic API to process XSQL Pages from within Java programs, now has a processToXML() method, allowing you to process the XSQL Page and get it's output as a DOM Document object, ready for subsequent processing by your program.

Process any XSQL page at the command line by using the XSQLCommandLine() class, sending the output to standard out or to a file, using the command:

```
java oracle.xml.xsql.XSQLCommandLine xsqlURI [outputFile] [param1=value1 ...
paramN=valueN]
```

The following files are provided to run the Command Line Utility:

- .\bin\xsql.bat DOS batch file
- ./bin/xsql shell script

These set up the CLASSPATH to contain all JAR files required (same ones listed for the Servlet above), and then calls JRE to run `oracle.xml.xsql.XSQLCommandLine()` class.

XSQLCommandLine Utility Examples

Try these XSQLCommandLine Utility example:

- To test the demo file /xsql/airport/airport.xsql type the following:

  ```
  xsql xsqlURI [outputFile] [param1=value1 ... paramN=valueN]
  ```

- To output results of processing airport.xsql to standard output type the following:

  ```
  xsql airport.xsql airport=sfo xml-stylesheet=none
  ```

- To output the results to the output.html type the following:

  ```
  xsql airport.xsql output.html airport=sfo
  ```

- To post an XML Document to an XSQL Page being processed by XSQLCommandLine utility, provide a URL or filename as the value of the posted-xml command-line parameter, such as:

  ```
  xsql insertnewsstory.xsql
  posted-xml=http://p.moreover.com/cgi-local/page?index_xml+xml
  ```

  This uses the XSQL Page named `insertnewsstory.xsql` to insert the XML contents retrieved by visiting the named URL.

XSQL Page Processor Usage

The XSQL Page Processor usage is described in the following sections:

XSQL Usage Overview

Figure 19–6 summarizes how the XSQL Page Processor is used.

The XSQL Page Processor can be accessed in three main ways:

- A — From a call or request in XSQL Servlet. This request can be made, for example, over the web using HTTP protocol.
- B — From the command line using the XSQL Command Line Utility
- C — From an application using the XSQLRequest() class. For example, <xsql:query>. This can be used in any command. It is a customizable Java program.

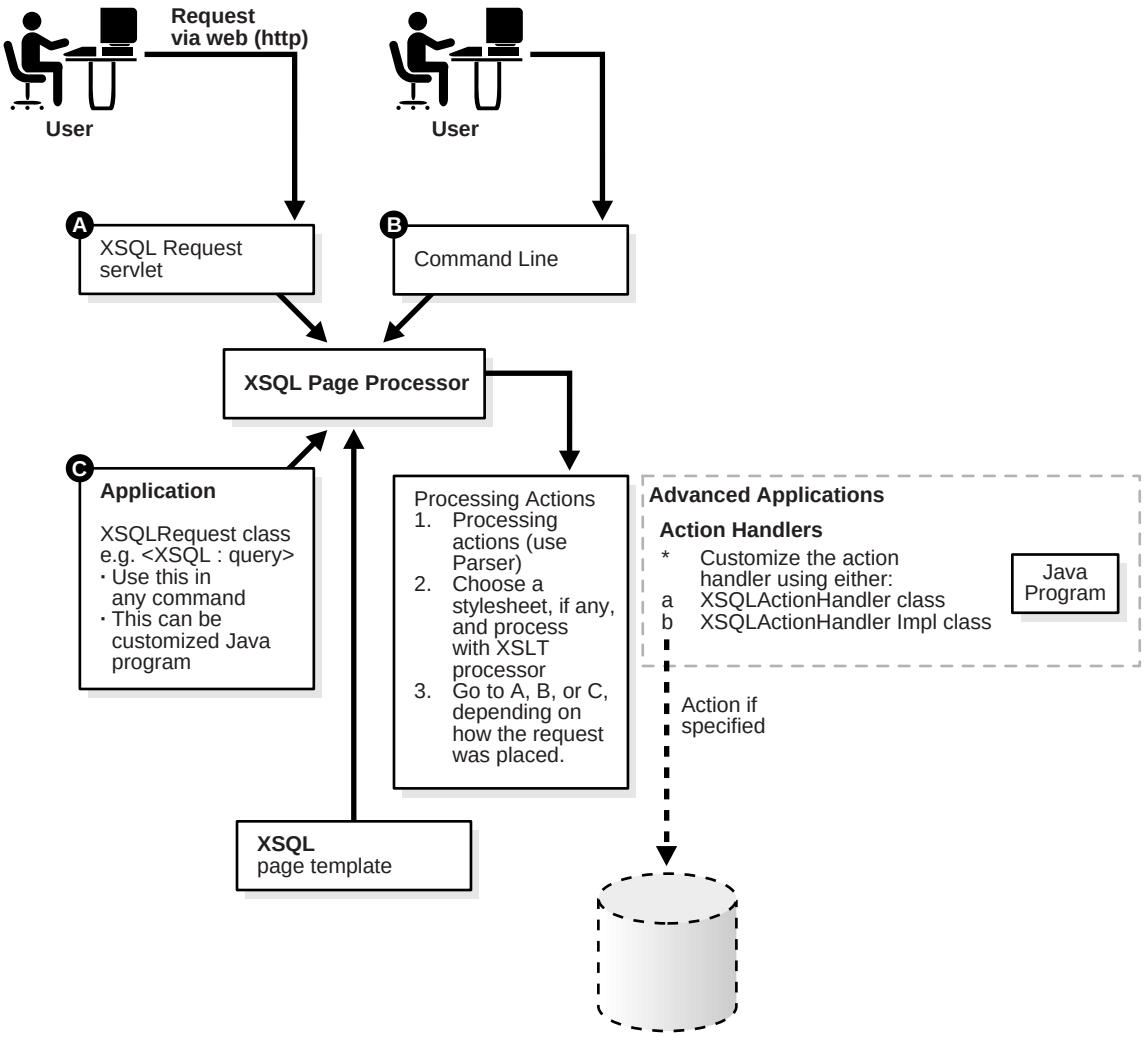
XSQL Page Processor processes the XSQL page template.

Processing actions are stipulated in the XSQL page template and processed by the XSQL Page Processor. The following transpires:

1. XSQL Page Processor determines (parses) the processing action to be carried out
2. A stylesheet, if any, is selected and processed using the XSLT Processor
3. The result or action is returned to A, B, or C, according to where the request came from.

4. You can also customize action handlers in Java using either of the following ActionHandler classes:
   - XSQLActionHandler() class
   - XSQLActionHandlerImpl() class

Figure 19–6 XSQL Page Processor: Overview



XSQL Page Processor Architecture

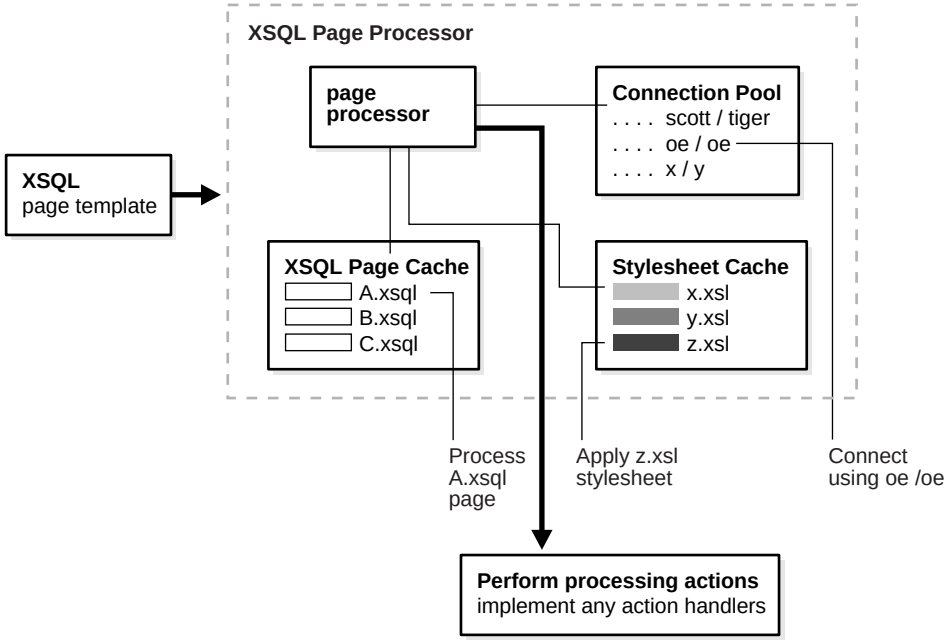
Figure 19–7 shows the main components in the XSQL Page Processor:

- Page Processor
- XSQL Page Cache
- Connection Pool
- Stylesheet Cache

When a XSQL Page Processor processes an XSQL page template, the Page Processor does the following:

1. Parses the templates, if new or changed.
2. Caches it.
3. Acquires an appropriate database connection from the Connection Pool.
4. Processes the actions specified on the template.
5. XSLT transforms it if appropriate.

Figure 19–7 XSQL Page Processor Architecture



XSQL Page Processor in Action

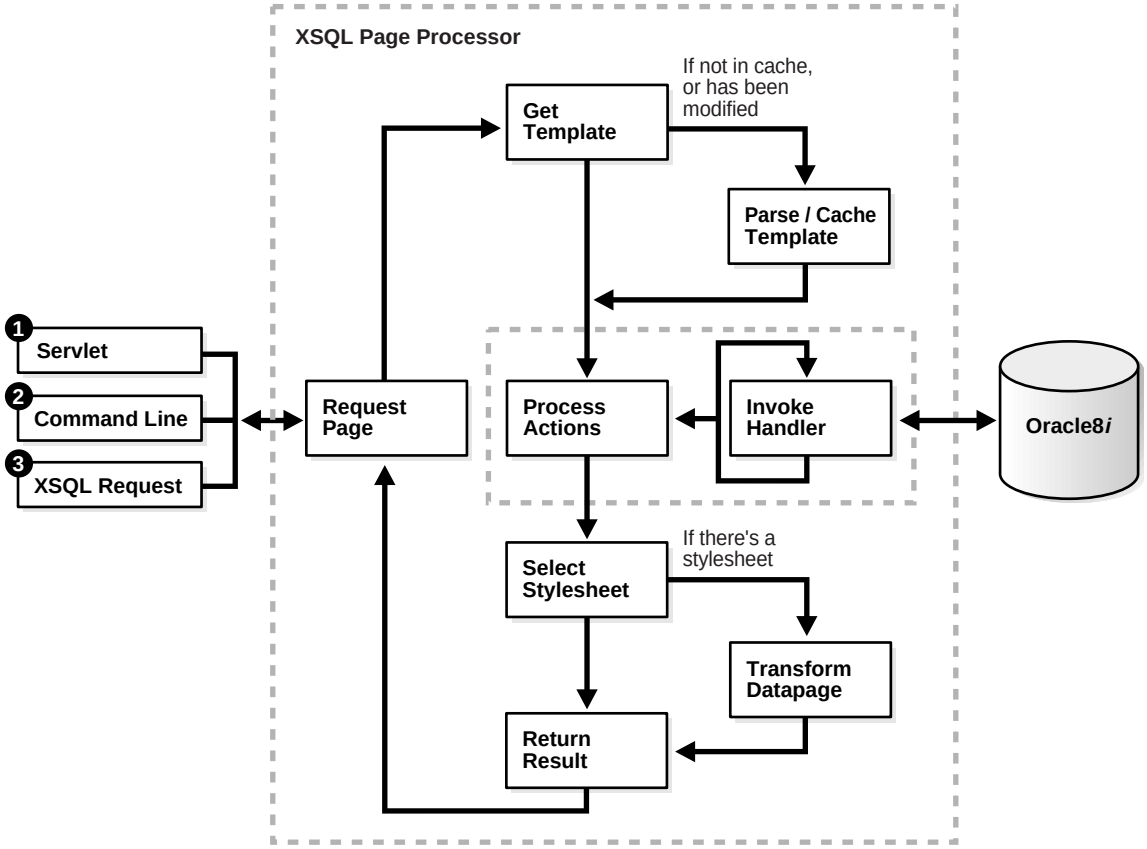
Figure 19–8 describes in detail what goes on inside the XSQL Page Processor when processing a request. See Figure 19–6 for the broad picture.

1. The XSQL Page Processor is accessed in any of the three ways shown:
   - From a Servlet
   - From the XSQL Command Line
   - From an XSQL Request

   The request page template input, "Request Page", is fetched by "get Template".
2. If the page is not in the XSQL Page Cache (See Figure 19–7, "XSQL Page Processor Architecture"), or if it has been modified, the request page is parsed and cached in the XSQL Page Cache for future use.

   If the page is already in the XSQL Page Cache it is simply fetched from the XQL Page Cache.
3. The results of the parse are processed for any actions or tasks, if specified in the XSQL Page Template and if applicable.
4. If the XSQL Page Template includes any ActionHandlers the Action Handler is invoked to further process the actions.
5. Next it applies the selected stylesheet and transforms the XSQL data Page.
6. The transformed result is returned to the requester, Servlet, XSQL Command Line, or XSQL Request application.

Figure 19–8 XSQL Page Processor in Action



XSQL Servlet Examples in demo/

Table 23–1 lists the XSQL Servlet example applications supplied with the software in demo/. Instructions for installing the demos are included in the Installation section of these release notes.

Table 19–3 XSQL Servlet Examples Supplied in demo/

Demonstration Name (File Name)	Description
Hello World (helloworld)	Simplest possible XSQL page.
Do You XML Site (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 (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 (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 (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 (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 19–3 XSQL Servlet Examples Supplied in demo/ (Cont.)

Demonstration Name (File Name)	Description
Airport Code Display (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 View 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 (adhocsql)	Shows how to pass an SQL query and XSLT Stylesheet to use as parameters to the server.
XML Document 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`.

Table 19–3 XSQL Servlet Examples Supplied in demo/ (Cont.)

Demonstration Name (File Name)	Description
	`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.
XML Insert Request 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 (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. Browse http://localhost/xsql/index.html to see a list of all the demos.

Setting Up the demo/ Data

To set up the demo data do the following:

1. Change directory to the .\xsql\demo directory on your machine.
2. In this directory, run SQLPLUS. Connect to your database as CTXSYS/CTXSYS (the schema owner for 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 ./xsql/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.

Using XSQL Servlet

Requirements

XSQL Servlet runs on any Java VM, using any JDBC driver, against any database.

XSQL Pages and XSQL Servlet support JDK 1.1.8 or JDK 1.2.2.

Note: JDK 1.1.7 has character set conversion UTF-8 routine problems that make it unusable for processing XSQL Pages.

Using XSQL Pages

Producing Dynamic XML Documents from SQL Queries with <xsql:query>

To use Oracle XSQL Pages for SQL queries, include <xsql:query> taga in your XML file wherever you want the SQL executed.

The <xsql:query> element is replaced by the XML output of your query. The XSQL Page Processor finds an attribute named connection on your XML document's element whose value must match the name of a connection defined in your XSQLConfig.xml file.

- *Include a Connection="conname" Attribute.* You must include a connection="connname" attribute on the document element of your XSQL page. If you do not, the XML file will be served to the requestor, but no <xsql:query> tags will be processed.
- *Connection name value must match XSQLCOnfig.xml.* Ensure that the value supplied for the connection name matches one of the entries in the XSQLConfig.xml configuration file in xsql/lib/.

A simple example using <xsql:query> tag is:

```
<?xml version="1.0"?>
<xsql:query xmlns:xsql="urn:oracle-xsql" connection="demo">
    SELECT 'Hello World' AS "GREETING" FROM DUAL
</xsql:query>
```

This produces the following XML document:

```
<?xml  version = '1.0'?>
  <ROWSET>
```

```
    <ROW id="1">
      <GREETING>Hello World</GREETING>
    </ROW>
  </ROWSET>
```

Any number pf <xsql:query> tags can be in an .xsql page. They can also be nested among other XML tags. But the .xsql must still be a well-formed XML document.

For example, you could build up a "data page" out of two queries like this:

```
<?xml version="1.0"?>
   <sales-by-year xmlns:xsql="urn:oracle-xsql" connection="salesdb">
     <period id="H1" year="CY99">
       <xsql:query>
         SELECT salesperson, SUM(sales) AS Total
           FROM sales
           WHERE sale_date between '01-JAN-99' and '30-JUN-99'
           GROUP BY salesperson
       </xsql:query>
     </period>
     <period id="H2" year="CY99">
       <xsql:query>
         SELECT salesperson, SUM(sales) AS Total
           FROM sales
           WHERE sale_date between '01-JUL-99' and '31-DEC-99'
           GROUP BY salesperson
       </xsql:query>
     </period>
   </sales-by-year>
```

This gives the following results:

```
<?xml version="1.0"?>
  <sales-by-year connection="salesdb">
     <period id="H1" year="CY99">
       <ROWSET>
         <ROW id="1">
           <SALESPERSON>Steve</SALESPERSON>
           <TOTAL>23465500</TOTAL>
         </ROW>
         <ROW id="2">
           <SALESPERSON>Mark</SALESPERSON>
           <TOTAL>39983400</TOTAL>
         </ROW>
       </ROWSET>
     </period>
```

```
    <period id="H2" year="CY99">
      <ROWSET>
        <ROW id="1">
          <SALESPERSON>Steve</SALESPERSON>
          <TOTAL>67788400</TOTAL>
        </ROW>
        <ROW id="2">
          <SALESPERSON>Mark</SALESPERSON>
          <TOTAL>55786990</TOTAL>
        </ROW>
      </ROWSET>
    </period>
  </sales-by-year>
```

You can customize many of the aspects of the XML query results produced for each <xsql:query> in your .xsql page by supplying one or more optional attributes on the appropriate <xsql:query> tag whose XML-results you'd like to affect.

Customizing Your Query with XML-SQL Utility Query Attribute Options

The following attribute-based options correspond to features offered by the underlying oracle.xml.sql.query, OracleXMLQuery class. This class is provided in Oracle XML-SQL Utility (XSU) for Java

Table 19–4 lists the <xsql:query> tag attributes supported by the XSQL Page Processor. These are case-sensitive.

Table 19–4 <xsql:query: Tag Attributes

Attribute Name	Description	
rowset-element	Element name to use for the query results. Set equal to the empty string to suppress printing a document element.	<ROWSET>
row-element	Element name to use for each row in the query results. Set equal to the empty string to suppress printing a row element.	<ROW>
max-rows	Maximum number of rows to fetch from the query. Useful for fetching the "top-N" or, in combination with skip-rows, the "next-N" rows from a query result.	fetch all rows
skip-rows	Number of rows to skip over before returning the query results.	not skip any rows

Table 19–4 <xsql:query: Tag Attributes

Attribute Name	Description	
id-attribute	Attribute name for the id attribute for each row in the query result.	id
id-attribute-column	Column name to use to supply the value of the id attribute for each row in the query result.	use the row count as the id attribute value
null-indicator	If set to y or yes, causes a null-indicator attribute to be used on the element for any column whose value is NULL.	omit the element in the result for any column with a NULL value tag-case

Built-in Action Handler

Table 19–2 lists the XSQL Pages built-in Action Elements.

Table 19–5 Action Handler Elements

Action Handler Element	Description
<xsql:query>	Execute an arbitrary SQL statement and include its result set in canonical XML format.
<xsql:dml>	Execute a SQL DML statement or PL/SQL anonymous block.
<xsql:set-stylesheet-param>	Set the value of a top-level XSLT stylesheet parameter. Value of the parameter can be set by supplying the optional "value" attribute, or by including a SQL statement as the element content.
<xsql:insert-request>	Insert the (optionally transformed) XML document that has been posted in the request into adatabase table or view. If HTML Form has been posted, then posted XML document is materialized from HTTP request parameters, cookies, and session variables.
<xsql:include-xml>	Include arbitrary XML resources at any point in your page by relative or absolute URL.
<xsql:include-request-params>	Include key information like HTTP Parameters, Session Variable values and Cookies into your XSQL Page for addressing them in your stylesheet.
<xsql:include-xsql>	Include the results of one XSQL Page at any point inside another.

Table 19–5 Action Handler Elements(Cont.)

Action Handler Element	Description
<xsql:include-owa>	Include the results of executing a stored procedure that makes use of the Oracle Web Agent (OWA) packages inside the database to generate XML.
<xsql:action>	Invoke a user-defined action handler, implemented in Java, for executing custom logic and including custom XML information into your XSQL Page.
<xsql:ref-cursor-function>	Includes the canonical XML representation of the result set of a cursor returned by a PL/SQL stored function.
<xsql:set-page-param>	Set a page-level (local) parameter that can be referred to in subsequent SQL statements in the page. The value can be set using a static value, the value of another parameter, or the results of a SQL statement.
<xsql:include-param>	Include a parameter and its value as an element in your XSQL page.
<xsql:set-session-param>	Set an HTTP-Session level parameter. The value can be set using a static value, the value of another parameter, or the results of a SQL statement.
<xsql:set-cookie>	Set an HTTP Cookie. The value can be set using a static value, the value of another parameter, or the results of a SQL statement.
<xsql:insert-param>	Inserts the value of a single parameter containing XML. Can optionally supply a transform to get it in canonical format.

See Also: the latest XSQL Servlet Release Notes at the OTN site, at http://technet.oracle.com/tech/xml, under XDK for Java > XSQL Servlet. The Release Notes document what additional actions you need to take if you are:

- Using the <xsql:include-owa> action element
- Running XSQL Servlet form inside a corporate firewall. Uncomment the http line...
- Using <xsql:include-request-params> element. Use specific format...
- Using <xsql:include-xml> element. Results must be well-formed.
- Using <xsql:insert-request> action element. First determine the canonical format you need for the inserts...

How XSQL Page Processor Processes ActionHandler Actions

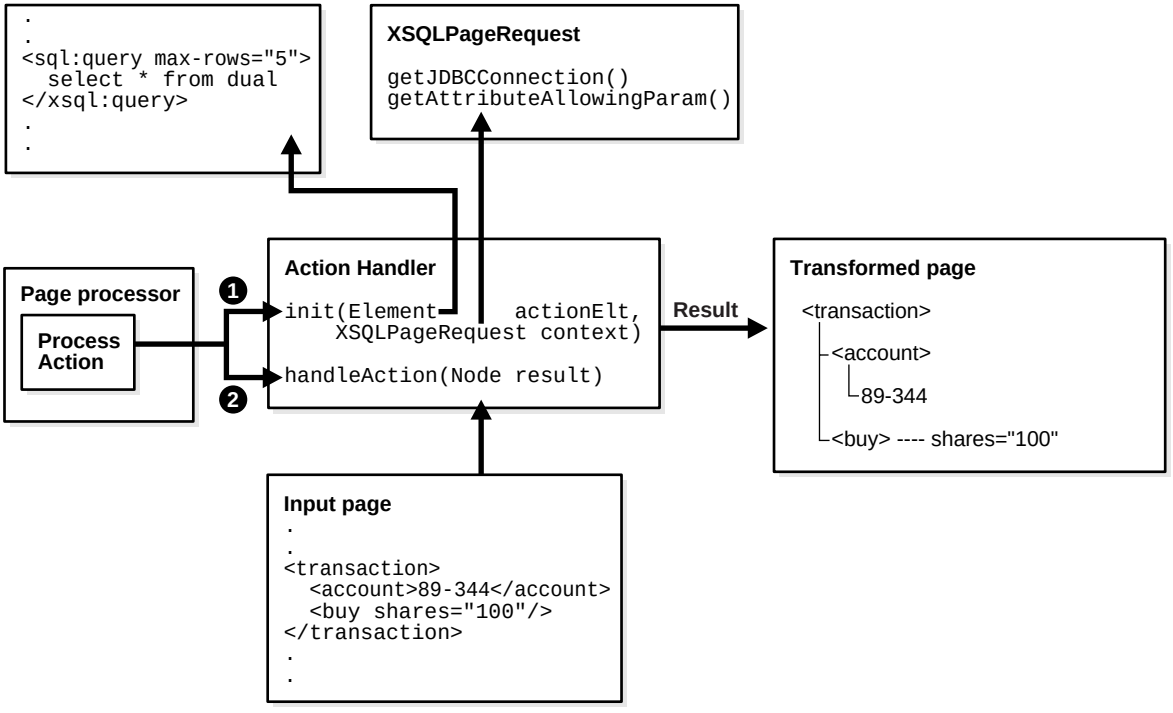
Figure 19–9 shows you how the XSQL Page Processor processes the actions specified in the ActionHandlers.

1. The XSQL Page Template inputs the ActionHandler.
2. The XSQL Page Processor's "process actions" module looks for two components in the ActionHandlers:
   - `init(Element actionElt,XSQLPageRequest context)`
   - `handleAction (Node result)`
3. `init(Element action Elt, XSQLPageRequest context)`
   - First parameter: Element contains for example, the SQL query such as:

     ```
     <xsql: query max-rows="5">
     SELECT * from dual
     </xsql:query>
     ```

   - Second parameter: XSQLPageRequest contains connection and page request information, such as getAttributeAllowingParam().
4. The Action Handler outputs the resulting transformed page

Figure 19–9 XSQL Page Processor: Processing Actions



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>
```

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>.

Using JavaServer Pages (JSP) and XSQL Pages

JavaServer Pages (JSP) can use <jsp:forward> to forward to an XSQL Page and <jsp:include> to include the results of an XSQL Page.

> **See Also:** *Oracle JavaServer Pages Developer's Guide and Reference* and the XML query example is in Chapter 9, "Sample Applications". There's also brief mention of OracleXMLQuery in Chapter 5, "Oracle Extensions", under "OracleJSP Support for XML and XSL".

Using <xsql:query> Tag Attributes

Attributes of the <xsql:query> tag (or any action element) can reference parameters in the same way the SQL statement for the tag can, allowing the client to pass in values for parameters like skip-rows or any other.

Client-overriding of stylesheets for an XSQL Page can be disallowed with the new allow-client-style="no" on the document element of the page.

Using the XSQL Page Processor Programmatically

`oracle.xml.xsql.XSQLRequest()` class uses the XSQL Page Processor from your Java programs.

For example:

```
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
    Example:

        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));
  }
}
```

You can also call `processToXML()` instead of `process()` to return the XML Document in-memory that results from processing the XSQL Page requested and applying any associated XSLT Transformation.

Customizing XSQL Action Handlers

See Figure 19–8, "XSQL Page Processor in Action" and Figure 19–9, "XSQL Page Processor: Processing Actions". XSQL Page Processor processes XSQL Pages by carrying out the following actions:

- Looks for Action Elements from the XSQL namespace.
- For each Action Element found, it invokes an appropriate action element handler class to process the element

Built-in XSQL Action Elements and Action Handler Classes

Table 19–6 shows the built-in action handlers provided with XSQL Pages for the following basic XSQL action elements.

Table 19–6 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-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

- Action Handlers are initialized by getting passed the XSQLPageRequest object for context and the root node of a DOM DocumentFragment to which the action handler should append any dynamically created elements or other DOM Nodes. The XSQL Page Processor replaces the action element in the template page with content of the DocumentFragment created by the appropriate Action Handler for the element.
- To create a custom Action Handler, you need to provide a class that implements the oracle.xml.xsql.XSQLActionHandler interface.

For convenience, you can save time by extending the base implementation class named `oracle.xml.xsql.XSQLActionHandlerImpl t`hat provides a default implementation of the init() method and offers a set of useful helper methods that should prove useful.

See Also: The XSQL Servlet Release Notes on OTN.

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);
      }
    }
```

Lastly, note that if you create custom action handlers that need to work differently based on whether the page is being requested through the XSQL Servlet, or 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 and response objects by:

```
XSQLServletPageRequest xspr = (XSQLServletPageRequest)getPageRequest();
     if (xspr.getRequestType().equals("Servlet")) {
       HttpServletRequest     req  = xspr.getHttpServletRequest();
       HttpServletResponse    resp  = xspr.getHttpServletResponse();
         // do something fun here with req and resp, however
         // writing to the response directly from a handler will
         // produce unexpected results. Allow the XSQL Servlet
         // to write to the servlet's response output stream
         // at the write moment later when all action elements
         // have been processed.
     }
```

Using a Custom XSQL Action Handler in an XSQL Page

To include an Action Element in your XSQL Page that invokes a custom Action Handler that is implemented by the yourpackage.YourCustomActionHandler class, add an element like:

```
<xsql:action handler="yourpackage.YourCustomHandler"/>
```

or

```
<xsql:action handler="yourpackage.YourCustomHandler" param="xxx"/>
```

or

```
<xsql:action handler="yourpackage.YourCustomHandler" param="xxx">
      Any text or <element>content</element>
</xsql:action>
```

Defining Custom XSQL Action Element for your Handler

For additional convenience, you can define entries in the XSQLConfig.xml file that associate an action element name and a handler class. So, for example, in the default XSQLConfig.xml file shipped with Oracle XSQL Pages, you'll see:

```
<actiondefs>
   <action>
     <elementname>param</elementname>
<handlerclass>oracle.xml.xsql.actions.ExampleGetParameterHandler</handlerclass>
     </action>
     <action>
       <elementname>current-date</elementname>
<handlerclass>oracle.xml.xsql.actions.ExampleCurrentDBDateHandler</handlerclass>
     </action>
     </actiondefs>
```

These entries in the config file allow a page to use:

```
<xsql:param name="myparam"/>
```

instead of the more verbose:

```
<xsql:action handler="oracle.xml.xsql.actions.ExampleGetParameterHandler"
                    name="myparam"/>
```

and similarly

```
<xsql:current-date/>
```

instead of the more verbose:

```
<xsql:action handler="oracle.xml.xsql.actions.ExampleDBDateHandler"/>
```

Using XSQLConfig.xml to Tune Your Environment

Use the XSQLConfig.xml File to tune your environment for working with XSQL Servlet. Parameters you can configure for your environment within the XSQLConfig.xml file include the following:

- The default number of records to fetch per database round-trip for <xsql:query> actions.
- Size (in pages) of the XSQL Page LRU cache.
- Size (in stylesheets) of the XSLT Stylesheet LRU cache.
- Size (in bytes) of the Buffered Output Stream to use for XSQL Servlet (or zero to disable)
- Initial and increment size of the connection pool, as well as the time-out threshold (in seconds) for cleaning up connections.
- Initial and increment size of the XSLT Stylesheet pool, as well as the time-out threshold (in seconds) for cleaning up stylesheet instances.
- MIME types for which to suppress setting character-set information.

Modifying XSQL Configuration Settings

XSQL Page Processor loads the first `XSQLConfig.xml` file that it can find in the CLASSPATH of the environment in which it is running. `XSQLConfig.xml` file contains configuration parameters that affect how XSQL Page Processor and XSQL Servlet work. Each configuration parameter is documented in comments before it in the XSQLConfig.xml file. By default, the XSQLConfig.xml file is located at ./xsql/lib.

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 outer CURSOR() functions return an empty rowset.

Frequently Asked Questions (FAQs) - XSQL Servlet

NoClassDefFoundError When Running XSQL Servlet Demos

Question

I downloaded the lastest version of the XSQL Servlet and the XML Parser for Java off of Technet and configured per the instructions. On both Java Web Server and JRun from Allaire, I get the following error (this one happens to be from JRun). As far as I know, I have the CLASSPATH set correctly, the xsql extension mapped to the servlet, etc.

```
(Running servlet) java.lang.NoClassDefFoundError:
oracle/xml/parser/v2/XSLException at
oracle.xml.xsql.XSQLServlet.doGet(XSQLServlet.java:118) at
javax.servlet.http.HttpServlet.service(HttpServlet.java:715)  at
javax.servlet.http.HttpServlet.service(HttpServlet.java:840) at
com.livesoftware.jrun.JRun.runServlet(JRun.java, Compiled Code) at
com.livesoftware.jrun.JRunGeneric.handleConnection(JRunGeneric.java:116) at
com.livesoftware.jrun.service.web.JRunWebServiceHandler.handleOutput(JRunWeb
ServiceHandler.java:266) at
com.livesoftware.jrun.service.web.JRunWebServiceHandler.handleRequest(JRunWebSer
viceHandler.java,
Compiled Code) at
com.livesoftware.jrun.service.ThreadConfigHandler.run(ThreadConfigHandler.java,
Compiled Code)
```

Answer

This is an error that the xmlparserv2.jar is not in your server side classpath. Double check that you've edited the CLASSPATH info in the right place. With JRun I do this from the JRun Admin application by...

1. On the (General) tab...
2. Click on the (Java) "sub"tab...
3. Enter the following into the "Java ClassPath" box (all on one line...)

   ```
   E:/xsql/lib/xmlparserv2.jar;
   E:/xsql/lib/oraclexmlsql.jar;
   E:/xsql/lib/oraclexsql.jar;
   E:/xsql/lib/classes111.zip;
   E:/xsql/lib;... etc ...
   ```

Specifying a DTD While Transforming XSQL Output to a WML Document

Question

I am trying to create a demo using XML's XSQL demo (downloaded from OTN). It works fine with Apache server. Now I am trying to write my own stylesheet for transforming XSQL output to WML and VML format. These programs (mobile phone simulators) need a WML document with a specific DTD assigned.

Is there any way, I can specify a particular DTD while transforming XSQL's output to a WML document.

Answer

Sure. The way you do it is using a built-in facility of the XSLT Stylesheet called <xsl:output/>

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output type="xml" doctype-system="your.dtd"/>
  <xsl:template match="/">
  </xsl:template>
     :
     :
</xsl:stylesheet>
```

This will produce an XML result with a:

<!DOCTYPE xxxx SYSTEM "your.dtd">

in the result. "your.dtd" can be any valid absolute or relative URL.

XSQL: Using the CURSOR Operator for Nested Structure

Question

1. The generated code has a top-level Element "<EMPLOYEES>". I want to get rid of it as per out requirements.
2. How can I specify id_attribute and id-attribute-column for the inner nested query e.g. <myEmployee myEmployee_id="1001">

Currently I am using "include-xsql" and manual deletion as a work-around. I believe most of the XSDs getting out of Oracle will be complex having nested structures. Just to give you an idea, I am working on an XSD such as the following:

```
Budget Worksheet
```

```
Positions
  Elements
    Element Distributions
  FTE Distributions
Accounts
```

Though work-arounds are available using XSLT, include-xsql, and so on, they involve additional work. Will XSQL support it?

Answer

1. The generated code has a top-level Element "<EMPLOYEES>". I want to get rid of it as per out requirements.

   Unfortunately, today we do not support renaming columns or attributes or hiding them directly with the utility. The only way to do it is to use an XSLT stylesheet (simple stylesheet which simply removes the EMPLOYEES tag).

2. How can I specify id_attribute and id-attribute-column for the inner nested query e.g. <myEmployee myEmployee_id="1001">

Again, we do not support this yet. You can do it with XSLT.

XSQL Servlet Conditional Statements

Question

Is it possible to write conditional statements in a .xsql file, and if yes what is the syntax to do that?

For example:

```
<xsql:choose>
    <xsql:when test="@security='admin'">
      <xsql:query>
          SELECT ....
      </xsql:query>
   </xsq:when>
   <xsql:when test="@security='user'">
      <xsql:query>
          SELECT ....
      </xsql:query>
   </xsql:when>
...
</xsql:if>
```

Answer

- I'd rather use <xsql:dml> to call a PLSQL procedure that would make the test. The statement you're describing look like xsl statement, used to process an already completed XML document. As far as I understand your problem, I would use a PLSQL procedure calling dbms_sql to generate the right statement, and perhaps go for a ref cursor.
- Right now, no, these are not supported. The plan for post-1.0.0.0 release is to integrate XSQL action handlers with XSLT so that you will be able to mix and match <xsql:query>, for example with XSLT tags like <xsl:choose> instead of copying all of XSLT's functionality with XSQL equivalents like <xsql:choose>.

We want to blend the best of both worlds and it will happen in a forthcoming release. For now, you can use the <xsql:ref-cursor-function> tag and make the conditional decision in the PL/SQL that returns the REF CURSOR.

Multiple Query in an .xsql File: Page Parameters

Question

I have two queries in an .xsql file.

```
<xsql:query>
   select col1,col2
   from table1
</xsql:query>
<xsql:query>
   select col3,col4 from table2
   where col3 = {@col1}    => the value of col1 in the previous query
</xsql:query>
```

How can I use, in the second query, the value of a select list item of the first query?

Answer

You do this with page parameters.

```
<page xmlns:xsql="urn:oracle-xsql" connection="demo">
  <!-- Value of page param "xxx" will be first column of first row -->
  <xsql:set-page-param name="xxx">
    select one from table1 where ...
  </xsl:set-param-param>
  <xsql:query>
     select col3,col4 from table2
```

```
    where col3 = {@xxx}
  </xsql:query>
</page>
```

XSQL, URN, xsl:script

Question

Using XSQL Servlet, I receive parameters, and I need to keep them safe for further use, in a few pages to be reached later. My idea is to keep them as hidden fields or the equivalent. I want to build an original XML document that looks like the following:

```
<root xmlns:xsql blah blah blah>
  <toBeTurnedAsForm action="doThis" method="POST">
    <aHiddenField name="secret" value="{@IGotYou}"/>
    ....
  </toBeTurnedAsForm>
</root>
```

As the <aHiddenField> tag does not belong to the XSQL name space, the parsing of the variable is not done. Is there a simple way to do it? I would not like to have a select '{@IGotYou}' from dual to make.

What does "urn" mean in the xmlns:xsql?

Answer

Use <xsql:include-param name="IGotYou"/> to include the value of a parameter in your datapage, then your XSL stylesheet can transform the:

<IGotYou>ValueOfIGotYou</IGotYou>

into wherever it needs to go in your HTML page.

urn stands for "Uniform Resource Name".

XSQL Servlet: Connecting to Any Database With JDBC Support

Question

Can the XSQL Servlet connect to any DB that has JDBC support.

Answer

Currently the XSQL Servlet is hardcoded for a connection to use OracleConnection class. We are testing a new version which would connect to any database.

XSQL Demos: Using the Correct Version JDK

Question

I'm trying to run the XSQL demo on Java WebServer. I have the classpath set and have mapped the .xsql extension to the XSQLServlet. When I go to the demo page and click on any of the links that point to an XSQL file, I get the following error message:

```
XSQL-001: Cannot locate requested XSQL file. Check the name.
```

I guess it is a mapping problem but I have mapped the extension. Also the Java WebServer console has this message: It can find the servlet)oracle.xml.xsql.XSQLServlet: init

Answer

Your problem may the location of the XSQL files. On my Apache install they are in ..htdocs/xsql/demo/ and the fully qualified path is properly registered.

I've set my classpath for Java Webserver 2.0 by creating a simple batch file which I used to start up the server. I've also aliased *.xsql to oracle.xml.xsql.XSQLServlet in the Servlet Aliases settings page.

Java Webserver also by default uses port 8080. Can you run any other servlet?

Comment

This happens to me too using Apache on Solaris. Some investigation reveals that this happens also when using xsqlcommandLine. Notice that the problem occurs Apache default web user) the command works perfectly under Oracle user or root.

It happens with JDK 1.1.5. JDK 1.1.7 and JDK 1.1.8 seem to work better. Now I have part of the solution but it does not resolve the demo connection, but I'll surely solve this too.

From XML To Table Creation?

Question

I created UNIV table as below. Whenever a new XML template file is created, I should create the new table schema as below:

```
CREATE TYPE student_type as object(
   id int,name varchar(20),
   math int,english int);/
CREATE TYPE department_type as object(
   name varchar(10),
   student student_type);/
CREATE TYPE college_type AS OBJECT(
   name VARCHAR(50),
   dept department_type);/
CREATE TABLE univ (college college_type);
```

Can the tables above be automatically created from the aproprietry XML file? In my project, the XML, DTD file is created by DTD Factory, which is a part of my project, dynamically. I need a solution that creates tables automatically from XML files.

Answer

You could use the XSLT processor in our XML Parser to generate the required SQL file for your table creation based upon the XML document.

Question 2

I need to create tables to import existing XML document. How can the table be automatically generated based on XML? I have download XML-SQL Utility, so I think I already have the XML Parser. How do I trigger this wonderful magic into action?

Answer 2

Download the XSQL Servelet archive as that has a number of demos that include the SQL scripts that map to XML documents. INSCLAIM.sql may be the most applicable as it use object views and several tables and should serve as a good template.

Question 3

Insclaim.sql is a far cry from the message which said, "You could use the XSLT processor in our XML Parser to generate the required SQL file for your table creation based upon the XML document."

If Insclaim.sql were created by hand, then it requires someone to create a structure upon structure to mimic the DTD. Are there alternatives?

I am running Microsoft IIS server 4.0 which does not support Java Servlet. Installing an Apache is not an option at this moment due to the time constraint to get a simple prototype done. For example, to import a set of valid XML into a repository and be able to search and apply a stylesheet upon retrieval. What do you suggest?

Answer 3

The previous reply about XSLT was based on the assumption that you were looking for something on an on-going basis, not simply something quick. The way to approach an automatic XSLT solution is the following:

- Have an idea of the mapping, hence my insclaim.sql response
- Create a simple SQL INSERT script based upon that mapping, by hand, with dummy data.
- After that works, substitute the XSLT instructions that retrieve your document data for the dummy data, thereby creating the stylesheet.

There is no quick automatic solution at this point because DTD's do not adequately describe the schema. XML Schema support however is forthcoming.

XSQL Servlet: Access to JServ Process

Question

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

and Oracle8i as database. I have Oracle client installed and Tnsnames.ora file configured to get database connection. My XSQconnections.xml file is configured correctly.

Answer

This looks like a generic JServ problem. You have to make sure that your security.allowedAddresses=property in jserv.properties allows your current host access to the JServ process where Java runs. Can you successfully run *any* JServ Servlet?

Comment

It works now. The JVM was not cleaning up properly when I stop Apache serv. After restarting every thing the XSQL servlet worked fine.

Calling xmlgen via XSQL Servlet on Java Web Server

Question

Now I want to call the xmlgen utility through my servlet which resides on Java Web Server 2.0. Can this be possible? Can I directly call the stored procedure executing the query which returns the XML document and directly host it on the browser through the servlet. If this is to be done should the browser be IE5.0

Answer

This is precisely what the XSQL servlet does. It is a servlet which uses the XML Parser for Java,V2 and the XML- SQL Utility to generate XML pages for SQL queries. Try using the XSQL servlet.

You can directly call the client side version of the XML-SQL Utility and the XML parser directly as well. If you are executing stored procedures which return the XML as a string, that would work as well.

The browser need not be IE5.0. The difference is that in IE5.0 if you supply a .xml file it would apply a default stylesheet and display it nicely. In other browsers this would only appear as a text file. If you apply XSL processing to the XML document and transform it in to HTML, then you can display it correctly on any browser.

Supporting Other Databases

Question

Does XSQL Servlet support SQL Server database? How about the other databases besides Oracle?

Answer

XSQL Servlet supports any database with a JDBC driver.

XSQL Servlet: Connecting to Remote Database

Question

I have successfully got the XSQL Servlet running on my NT system (Oracle*8i*, Apache Web Server, IE5) but I cannot run it under Solaris. My Solaris system does not as yet have Oracle*8i* installed and therefore the XSQL Servlet needs to connect to the remote Oracle*8i* on my NT machine - and I think this is where the problem lies.

I have installed on the Solaris system Oracle 8.0.5, Apache 1.6.3, ApacheJserv1.0, and the UNIX installation of XSQL Servlet.

The test of the ApacheJserv works correctly. The error messages I get when trying to access the xsql_servlet demo pages are:

```
In the err_log file...[Wed Aug  4 11:21:20 1999] [notice] Apache/1.3.6 (Unix)
ApacheJServ/1.0 configured -- resuming normal operationsCould not establish the
JDBC connection:java.sql.SQLException: No more data to read from socketand in
the mod_jserv.log file....[04/08/1999 12:03:55:536] (ERROR) ajp11: Servlet
Error: java.lang.NullPointerException: null[04/08/1999 12:03:55:536] (ERROR) an
error returned handling request via protocol "ajpv11"
```

My XSQLConnections.xml file contains the following:

```
... scott    tiger    jdbc:oracle:thin:@pc2929:1521:peter        xmldemo
xmldemo    jdbc:oracle:thin:@pc2929:1521:peter
```

and I have also tried setting the dburl to:

```
jdbc:oracle:thin:@(description=(address=(protocol=tcp)(host=pc2929)(port=1521))(
source_route=yes)(connect_data=(sid=peter)))
```

The connection to the remote database (peter) functions correctly if I use SQL*PLUS from the Sun. What could be wrong? Is it simply that the database must be on the same machine as the WEB server?

Answer

It is designed to work anywhere JDBC can connect to. It definitely does not require the database to be on the same machine as the web server. But, it *does* require that you can successfully establish a JDBC connection using the JDBC driver that it finds in the Apache JServ's CLASSPATH environment, given the connect information in the XSQLConnections.xml file.

Most problems crop up because the CLASSPATH for Jserv does not properly include the JDBC classes111.zip.

I'd suggest the following:

1. Try a JDBC sample program with the connect string you have in XSQLConnections.xml to make sure you can connect at all
2. Check the JServ classpath for the correct inclusion of the latest Oracle JDBC driver archive file.

Comment

The problem was solved by installing the newest 8.05 JDBC libraries on the Solaris machine.

Apache JServ

Question

I've set up Apache (1.3.9) + JServ + XSQL Servlet as explained in the XSQL servlet release notes. When I try to run: http://127.0.0.1/xsql/demo/helloworld.xsqlI see the following entry in mod_jserv.log :[11/10/1999 20:25:47:383] (ERROR) ajp12: Servlet Error: ClassNotFoundException: oracle.xml.xsql.XSQLServlet

Does anybody know why JServ cannot find the class above? 'IsItWorking' example seems to work ok, so Apache is talking to JServ.

```
relevant (?) stuff from jserv.properties:wrapper.bin=c:\jdk1.2.2\bin\java.exe
wrapper.classpath=C:\Program Files\Apache Group\Apache
JServ\ApacheJServ.jarwrapper.classpath=c:\jsdk2.0\lib\jsdk.jarwrapper.classpath=
C:\xsql\lib\classes111.zipwrapper.classpath=C:\xsql\lib\xmlparser.jarwrapper.cla
sspath=C:\xsql\lib\oraclexmlsql.jarrelevant (?) stuff from mod_
```

```
jserv.conf:ApJServMount /servlets /rootApJServAction .xsql
/servlets/oracle.xml.xsql.XSQLServlet
```

Answer

```
wrapper.classpath=c:\jsdk2.0\lib\jsdk.jar: :
wrapper.classpath=C:\xsql\lib\classes111.zip: :
wrapper.classpath=C:\xsql\lib\xmlparser.jar: :
wrapper.classpath=C:\xsql\lib\oraclexmlsql.jarYou are
missing:wrapper.classpath=C:\xsql\lib\oraclexsql.jar
```

This is the jar file for XSQL Servlet itself.

Document Creation

Question

When I install XSQL Servlet and run the query it should create a XML document on the fly. How the document is being created. As far as my understanding goes we need to hardcode the document. Explain me how it's being created automatically. In that case where do I keep the DTD. I am not seeing any XML documents embedded with DTD. Where does the parser come to picture? Do I need to install any tools to create the documents.

```
SELECT salesperson, SUM(sales) AS Total
   FROM sales
   WHERE sale_date between '01-JAN-99' and '30-JUN-99'
   GROUP BY salesperson
SELECT salesperson, SUM(sales) AS Total
FROM sales
       WHERE sale_date between '01-JUL-99' and '31-DEC-99'
     GROUP BY salesperson
```

which would produce results like:

**** How is this being created automatically?****

```
 Steve        23465500                  Mark        39983400
 Steve        67788400                  Mark        55786990
```

Answer

XSQL Page Processor (used by the XSQL Servlet) processes your page looking for tags. Whenever it finds one, it creates a new instance of an OracleXMLQueryobject (part of the Oracle XML SQL Utility for Java, also downloadable separately) and

asks *that* utility to produce the XML for the currentquery. Everything you need to run XSQL Servlet comes bundled with it in it's distribution and it's release notes details the libraries it depends on.

XSQL with JRUN

Question

I'm trying to use XSQL with JRUN on IIS4.0 and getting:XSQL-007: Cannot acquire a database connection to process pageConnection refused(DESCRIPTION=(TMP=)(VSNUM=135286784).....when I execute: xsql emp.xsqlI've tested my JDBC with samples in Oracle8i's JDBC directory and all seems to working fine. I've also tested my JRUN environment running other servlets. I'm also able to connect using SQLW.It maybe the XSQLConnections.xml file. I have it at:jsm-default\services\jws\htdocs\ with the startup index.html file. I'm also using JRUN's default 8000 port if that matters.Is there anything else I should have done or I could test to ensure that XSQL can connect to Oracle8i?

Answer

It's definitely finding the XSQLConnections.xml(you would have gotten a different error if that were the case), it's just failing to get the JDBC connection using the connection info in your file.What JDBC driver are you using in your XSQLConnections.xml file? If it's not the "thin" driver, are the appropriate paths do necessary DLL's setup correctly?

XSQL on Oracle8i Lite

Question

I am trying to use XSQL with Oracle8i Lite (Win 98) and Apache/JServ Webserver. I am getting error message "no oljdbc40 in java.library.path" even though I have set the olite40.jar in my classpath (which contains the POLJDBC driver). Is there anything extra I need to do to run XSQL for Oracle8i Lite.

Answer

You must include the following instruction in your

```
.\jserv\conf\jserv.properties file:wrapper.path=C:\orant\bin
```

where C:\orant\bin is the directory where (by default) the OLJDBC40.DLL lives.

NOTE this is *NOT* wrapper.classpath,it’s wrapper.path.

xsql:ora-uri tag

Question

I was just viewing XSQL demo codes and came across tag which I can't find in the release note.Could you tell me what it does and its attributes?Thanks,H.Ozawa

Answer

<xsql:ora-uri>

is one of several demonstrations of the extensible action elements that will be available in the 0.9.8.3 release of Oracle XSQL Servlet.It is an undocumented test tag in thisrelease.As you have been helping us test the pre-release version you already have the upcoming release in house.For others, it will be available shortly, with lots of new functionality.

Multiple Parameters in the XSQL Servlet

Question

I have a question about XSQL Servlet.Is there any way to handle multiple <form> parameters with the one same name, which is needed for <input type="checkbox">?

Answer

XSQL is just a shell to send SQL request. You'll still need to write a logic to convert HTML to request to SQL.

I'll probably use the NAME attribute in INPUT tag to differentiate selections and use it in the WHERE clause.

XSQL Servlet and Oracle 7.3

Question

Is there anything that prevents me from running the XSQL Servlet with Oracle 7.3? I know the XML SQL Utility can be used with Oracle 7.3 as long as I use it as a client-side utility.

Answer

Nothing inherent in the XSQL Servlet prevents it from working against 7.3.You just need to make sure: - Your servlet runner environment has an appropriate JDBC driver in the CLASSPATH (if the default 8.1.5 driver doesn't work, which it should), and - That your XSQLConnections.xml file properly reflects the <dburl> that you need to connect to the DB you want to.

Passing Parameters Between XSQL and XSL

Question

I am trying to bring up a form p2.xsql with n no. of records based on a value of (@pname) passed from form p1.xsql. Now say this returns more than 100 records but I am viewing only 10 at a time with previous and next tags, on the click of next I want to fetch the another set of 10 records from the list for @pname, But I am not able to pass the value of pname here, how can I do the same, say on the href of <next> I want to pass pname as a name value parameter.

Answer

Did you try using 'max-rows' and 'skip-rows '?Set 'max-rows' to 10 and keep @pname and variables for 'skip-rows' as a global or static variable in your client side program.You'll just have to increase or decrease the value of 'skip-rows' variable which you will be passing to your servlet.

The latest release contains several new features that make this easier, but in release 0.9.6.2, the simplest way to get a parameter value into your datapage is to do:

```
<query rowset-element="" row-element="">
select '{@pname}' as pname from dual
</query>
```

Conditional Query

Question

Is it possible for me to put any conditional query in the .xsql file itself?

For example: If I have three parameters, depending on what is populated, I want the query string to be formed. Is it possible for example to state,

```
if(p1=NULL) <qurey1>.....</query1>
if (p2=NULL) <query1>.....</query2>
```

and so on?

Answer

A better way is to do a selection on a client side using Javascript. XSQL is not a language so it is very limited in terms of selection and looping.

Are the three different choices radically different, or slight variations of WHERE clauses against the same table? If the latter, you can do the following:

```
SELECT ....  FROM TABLE  WHERE ( ({@p1}='yes') and (...something...))
```

or

```
( {{@p2}='yes') and (...something else...))
```

XSQL Servlet and INSERTs or UPDATEs

Question

Can we perform DML operations using XSQL servlet? We can select from tables and present it in XML or HTML format. Can we do an Insert or Update using the servlet?

Answer

From release 0.9.8.6 XSQL Servlet supports this, along with other new enhancements, using the <xsql:dml> tag.

XSQL Servlet and Dynamic SQL

Question

Does XSQL Servlet support dynamic SQL? Take a stock screening page as an example. In this example you want to dynamically build the where clause based on user input. Thus if a Minimum PE Ratio of 3 was entered, the where clause of "where PE_Ratio >= 3" would be appended to the query. In this case there may be up to 20 different parameters or more to be mapped dynamically, so it wouldn't be feasible to nest all of the different combinations.

Answer

XSQL supports lexical substitution parameters so any and every part of any query can be parameterized. The extreme case is:<query> {@sql} </query>, where the entire query is passed in a parameter (named "sql"). But any combination of parameters and substitutions is legal and can be used to do what you want. Since the variables are not *BIND*variabled, but instead lexical substitution variables, you can do things like:

```
<query>
  select {@collist} from {@table} where {@where}
  order by {@orderby}
</query>
```

You can provide default parameter values using XML attributes on the <query> element which then can be overridden if a value is passed in the request

```
...<query collist="ename, sal"
     orderby="sal desc" where="1=1"       from="dept">
     select {@collist}     from {@table}     where {@where}
     order by {@orderby}
   </query>
```

And then a request to the page can pass in a orderby=decode(comm,null,1,0) or something to override the defaults.

XSQL Servlet and Other Relational Databases

Question

Does XSQL Servlets interact with other relational databases?

Answer

Yes. You need to provide the following:

- Appropriate JDBC driver in your CLASSPATH
- Appropriate <driver> and <dburl> settings for the connection in your XSQLConnections.xml file

Out Variable Not Supported in <XSQL:DML>

Question

I'm trying to determine the use of **<xsql:dml>**. I tried calling a stored procedure which has one out parameter, but I was not able to see any results. The executed code resulted in the following statement:

```
<xsql-status action="xsql:dml" rows="0"/>.
```

Answer

OUT variables are not supported in this release with <xsql:dml>, just IN parameters. This is because the {@param} variables in XSQL are *lexical* variables and not bind variables.Tell me more about what you're trying to do with the OUT value and I can likely suggest a different approach to achieve the same result. Are you trying to return an OUT value of a stored procedure into your XSQL "datapage"?

For an example, see: ./xsql/test/dml-test.xsql in the latest release.

Question 2

That what I'm doing....getting the out variable value onto the XSQL page? Is there a work around?

Answer 2

I think the simplest workaround would be to wrap your procedure with a function and then use an **<xsql:query>** to SELECT the function value as part of a SELECT statement.

So, assume your procedure looked like:

```
procedure foo( y number, x out number );
```

You can build a function that looks like the following:

```
function foowrapper( y number) return number is x number;
begin foo(y,x);
return x;
end;.
```

Then use an **<xsql:query>** action element to do the following:

```
<xsql:query>  select foowrapper({@yparam}) from dual</xsql:query>
```

Running XSQL Page Processor on Java Web Server?

Question

Can XSQL page processor servlet run on JavaWeb Server? The installation notes say that it has been tested with Apache web server. If I can run on Java Web Server, are there any pointers?

Answer

Since XSQL is a servlet, it can be run on any servlet engine and that includes Java Web Server. Although I haven't worked on Java Web Server, I think there must be a mechanism to associate a handler for .xsql pages to the XSQL servlet. Once this is done I think Java Web Server should take XSQL requests.

The Release Notes document what web servers we've tested it with, although any Servlet Engine that's faithful to the Servlet specification should do.

Sun's Java Server Web Development Kit (JSWDK 1.0) is one of the servers we've tested it with. The only basic steps to installation (whose details might differ for *your* servlet engine) are:

- Make sure all of the following zips, archives, directories are in your servlet engine's CLASSPATH:
  - ./xsql/lib/xmlparserv2.jar
  - ./xsql/lib/oraclexmlsql.jar
  - ./xsql/lib/oraclexsql.jar
  - ./xsql/lib/classes111.zip
  - ./xsql/lib
- Map the extension .xsql to the Servlet class oracle.xml.xsql.XSQLServlet

SID and JDK Errors

Question

Don't know where exactly to put the config file - using Websphere 3. Also, I get the following error after 'setting everything up' and trying to access the helloworld.xsql:

```
Oracle XSQL Servlet Page Processor 0.9.8.6 (Technology Preview)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 racle:thin:@localhost:1521:ORCL</dburl>
<driver>oracle.jdbc.driver.OracleDriver</driver>
    </connection>
```

So the error you're getting is likely because:

1. Your database is not on "localhost" machine.
2. Your database SID is not ORCL
3. Your TNS Listener Port is not 1521

Make sure those values are appropriate for your database and you should be in business.

Question 2

Good stuff it now sort of works. The SID was wrong.

Using Custom Action Elements: XSQLActionHandlerImpl

Question

Can I pass parameters to my Java class which extends XSQLActionHandlerImpl.

Where is there extensive documentation for XSQL?

Answer

The Release Notes and the JavaDoc fully document how to use the custom Action Elements. The distribution comes with source code for two sample Action Elements, too.

./xsql/src/oracle/xml/xsql/actions/ExampleGetParameterHandler.java
./xsql/src/oracle/xml/xsql/actions/GetCurrentDBDateHandler.java

These illustrate how an Action Element uses its init() method to pick up parameters from any attributes or nested elements of the action element in the XSQL Page and how they write output to the resulting page. You shouldn't have a need to pass parameters to the Action Element Handler, any information should be picked up from attributes or elements in the XSQL Page.

For example:

```
<xsql:action handler="my.pkg.SampleHandler"    arg1="foo" arg2="bar">
    <statement> 1,2,3,4,5 </statement>
 </xsql:action>
```

Your action handler can use standardDOM techniques in its init() method (or make use of some of the helper methods provided in XSQLActionHandlerImpl like getAttributeAllowingParam() to pickup parametrized values to drive the action.

XSQL Servlet: Writing an Action Element (Handler) to Set a Browser Cookie

Question

There is an application, where some users login, and do some operations, say some inserts/updates, whenever there is any such DML statement performed on the database, except of course SELECT, I want a field called the user_id to be updated with the user who had done it and also the time it was changed.

My application is basic, a set of .xsql files. My webserver is Apache Web Server. I am using Java Servlet (jrun). I have the latest XSQL Servlet set up.

Where is such an array of structures with user profiles, like user_id/name, ip address of connect from, and so on is stored. For example, is an environmental variable set somewhere?

Answer

If you look at environment variables, they will be the ones on the web server machine and likely not what you want. When the user logs-in to your application,

you could write a small XSQL custom action element to set a browser cookie to the value of their username, and then refer to that cookie value for the value of USER_ID in your database inserts.

Question 2

I think storing it on the cookie and using it will be perfect. I do not know how to set the browser cookie, is there any an example to which I could refer?

I found couple of things such as, set Cookie:Name=....., and something on owa_Cookie. Which should I use with XSQL?

Answer 2

You'll need an action handler that does the following:

```
:import javax.servlet.http.*;
:XSQLServletPageRequest xspr = (XSQLServletPageRequest)getPageRequest();
if (xspr.getRequestType().equals("Servlet"))
 {
  HttpServletResponse resp  = xspr.getHttpServletResponse();
Cookie newCookie = new Cookie("name","value");
resp.addCookie(newCookie);
}
   }
```

Installing XSQL Servlet on IIS with JRun

Question

My web server is IIS with JRun. How do I install the XSQL servlet?

Answer

Ensure, using the appropriate place in the JRun administration utility, that you have the following in your servlet engine classpath. Assuming you've installed XSQL in E:

```
\ ...
E:\xsql\lib\classes111.zip
E:\xsql\lib\oraclexmlsql.jar
E:\xsql\lib\oraclexsql.jar
E:\xsql\lib\xmlparserv2.jar
E:\xsql\lib
```

Ensure that you make the .xsql extension to the oracle.xml.xsql.XSQLServlet servlet class.

Writing an XSQL Action Handler to Acquire HTTP Request Parameters

Question

I am trying to make this work by either using cookie and or by using the ENV structure stored at the browser end. With the cookies I am clear how to set and get values. I want this option in case somebody tries to disable cookie on their browser and it may not work. I want the environment variables to be available. How can this be done?

Using a Perl script, I can do a get of the ENV structure / array and use the same. How can I get the same functionality with XML and XSQL?

Answer

I don't understand how checking the system environment variables of the *server* machine can function as a fallback for a cookie to identify the user. At best, they would identify the user name of the user that is running the web server executable, not the current user logged into the browser.

If you can access the information you're looking for in Java then you can write an XSQL Action Handler to retrieve it. There's nothing built-in to handle this.

If by "environmental variables from the browser" you simply mean HTTP request parameters, then you can look at the example Action Handler provided with XSQL that illustrates how to get the HTTP request parameters for an idea of how to implement something similar.

Converting HTML Key Value Pairs to XML: <xsql:insert-request>

Question

How do I use the <xsql:insert-request> tag? I want to convert the key value pairs in a HTML form to XML. I understand that using XSQL Servlet and <xsql:insert-request> make it easier to do.

I need to save the converted XML file in a directory. Here is the form:

```
<html>
<body>  Insert a new news story...  <form action="form.xsql" method="post">
<b>Title</b>
```

```
<input type="text" name="title_field" size="30">
<br>
    <b>URL</b>
<input type="text" name="url_field" size="30">
<br>    <br>    <input type="submit">
  </form>
<body>
</html>
```

and here is the form.xsql (I am stuck here!)

```
 <?xml version = '1.0'?>
<xsql:insert-request-params transform=????/>
```

How does one just save the converted XML file? Can you suggest how to pass on the XML file to another servlet for processing, either saving it or without saving it?

Answer

<xsql:insert-request> is designed to insert the posted XML document (or posted HTML form parameters synthesized into an XML document) into the database. If you want to save it as a file in the server, then you can write a custom action handler (see the Release Notes) which calls:

```
getPageRequest().getPostedDocument()
```

And then writes it to a file by creating a new FileWriter()

Should be just a few lines of Java. The hardest part will be deciding what file name you want to give to the file.

Running a Procedure from the .xsql File

Question

Here is the error message:

```
- <test>- <xsql-error action="xsql:include-owa">  <statement>declare buf
htp.htbuf_arr; param_names owa.vc_arr; param_values owa.vc_arr; rows integer :=
32767; outclob CLOB;begin param_names(1) := 'HTTP_COOKIE'; param_values
(1) := ''; param_names(2) := 'SERVER_NAME'; param_values
(2) := 'shanthi.ijs.com'; param_names
...
```

My .xsql file is:

```
<test connection="system" xmlns:xsql="urn:oracle-xsql">
<
xsql:include-owa>
scott.get_date;<
/xsql:include-owa><
/test>
```

Answer

Seems like your DBMS_LOB package is either not installed or not installed correctly. Try re-running DBMSLOB.SQL in ./rdbms/admin of your Oracle installation.

XSQL File Launching from JDeveloper

Question

Is it possible to use the WebToGo that comes with JDev481 to test XSQL files. If so how what kind of configuration do I need to do.

Answer

This is the .bat file that I use "wtg.bat". It sets up the WebToGo webserver with the jar's for Oracle Lite, WebToGo, and the XSQL Servlet.

This assumes you have JDeveloper installed in J:\

The System property settings are so that the JDK URL classes can "see through" firewalls properly (for example, when parsing an XML document which has a DTD whose SYSTEM identifier is http://somesiteoutside.com/foo.dtd)

```
@echo off
set WTGCLASSPATH=j:\java\lib\classes.zip
set WTGCLASSPATH=%WTGCLASSPATH%;C:\orant\lite\classes\oljdk11.jar
set WTGCLASSPATH=%WTGCLASSPATH%;C:\orant\lite\classes\olite40.jar
set WTGCLASSPATH=%WTGCLASSPATH%;J:\lib\webtogo.jar
set WTGCLASSPATH=%WTGCLASSPATH%;J:\lib\xmlparser.jar
set WTGCLASSPATH=%WTGCLASSPATH%;J:\lib\classgen.jar
set WTGCLASSPATH=%WTGCLASSPATH%;J:\lib\servlet.jar
set WTGCLASSPATH=%WTGCLASSPATH%;J:\lib\ojsp.jar
set WTGCLASSPATH=%WTGCLASSPATH%;c:\xsql\lib
set WTGCLASSPATH=%WTGCLASSPATH%;c:\xsql\lib\classes111.zip
set WTGCLASSPATH=%WTGCLASSPATH%;c:\xsql\lib\oraclexsql.jar
set WTGCLASSPATH=%WTGCLASSPATH%;c:\xsql\lib\oraclexmlsql.jar
set WTGCLASSPATH=%WTGCLASSPATH%;c:\xsql\lib\xmlparserv2.jar
```

```
echo %WTGCLASSPATH%
jre -DproxySet=true -DproxyHost=www-proxy.us.oracle.com -DproxyPort=80 -clas
spath %WTGCLASSPATH% oracle.lite.web.JupServer
```

This is my J:\lib\webtogo.ora file:

Notice the ROOT_DIR line and the xsql= line.

```
[WEBTOGO]
PORT = 7070
USE_SYSTEM_CLASSPATH=YES
DEBUG=YES

[FILESYSTEM]
TYPE=OS
ROOT_DIR=C:\

[MIMES]
html=text/html
jsp=text/html;handler=oracle.jsp.runner.JspRunner
xsql=text/html;handler=oracle.xml.xsql.XSQLServlet

[APPLICATIONS]
xmlfile=.\wtgapp.xml

[SERVLET_PARAMETERS]
```

Can I Load Multiple Forms to My Database Using XML?

Question

I'm currently running Oracle Forms on the web. Will XML allow me to create a real-time interface whereby I can load multiple records into my Oracle database without going through a form?

Answer

Sure. Check out our XSQL Servlet at: http://technet.oracle.com/tech/xml/xsql_servlet. It comes with features, documentation, and demos to easily work with SQL, XML, and XSLT, including the ability to help in posting HTML Forms and posted XML documents into the database.

Try out the demos online at:
http://technet.oracle.com/tech/xml/demo/demo1.htm

In particular the "XML Insert Request Demo" shows a simple example of what can be done by posting XML to your database.

Getting and Storing XML Documents in a Database: Testing Functionality

Question

I downloaded the XML-SQL Utility, XSQL Servlet, Apache Webserver, Apache Jserv, and so on onto Windows NT. I installed them according to respective installation instructions. How do I test the functionality to get and store XML documents into database? And how should I test that all the above mentioned software installed are communicating correctly. I am using Oracle8 client on NT and Oracle8i server on Sun Solaris.

Answer

XSQL Servlet comes with a number of demonstrations to help test if it is properly installed. The release notes explain how to set them up. You can see the exact same set of demos at: http:/ /technet.oracle.com/tech/xml/demo/demo1.htm To compare results.

20

Using XML Transviewer Beans

This chapter contains the following sections:

Accessing Oracle XML Transviewer Beans

The Oracle XML Transviewer Beans are provided with Oracle8i Enterprise and Standard Editions from Release 2 (8.1.6). If you do not have these editions you can download the beans from the following site:

http://technet.oracle.com/tech/xml

XDK for Java: XML Transviewer Bean Features

XML Transviewer Beans facilitate the addition of graphical or visual interfaces to your XML applications.

Direct Access from JDeveloper

Bean encapsulation includes documentation and descriptors that can be accessed directly from Java Integrated Development Environments like JDeveloper.

Sample Transviewer Bean Application is Included

A sample application that demonstrates all of the beans to create a simple XML editor and XSL transformer is included with your software.

The included sample application with XML-SQL Utility (XSU) cause the following:

- Database queries to materialize XML
- Transform the XML through an XSL stylesheet
- Store the resulting XML document back in the database for fast retrieval

Database Connectivity

Database Connectivity is included with the XML Transviewer Beans. The beans can now connect directly to a JDBC-enabled database to retrieve and store XML and XSL files.

XML Transviewer Beans

XML Transviewer Beans are comprised of the following five beans:

DOM Builder Bean

The DOM Builder Bean is a non visual bean. It builds a DOMTree from an XML document.

DOM Builder Bean encapsulates the XML Parser for Java's DOMParser class with a bean interface, and extends its functionality to permit asynchronous parsing. By registering a listener, Java applications can parse large or successive documents having control return immediately to the caller. See "Using DOMBuilder Bean (Async API)" on page 20-5.

XSL Transformer Bean

The XSL Transformer 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.

XSL Transformer 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, XSL Transformer Bean enables an application or user to view the results of transformations immediately.
- This bean can also be used as the basis of a server-side application or servlet to render an XML document, such as an XML representation of a query result, into HTML for display in a browser.

See "Using XSL Transformer Bean (oracle.xml.async API)" on page 20-10.

Tree Viewer Bean

The Tree Viewer Bean displays XML formatted files graphically as a tree. The branches and leaves of this tree can be manipulated with a mouse. See "Using Treeviewer Bean (XMLTreeView() API)" on page 20-14.

XML Source Viewer Bean

The XML Source Viewer Bean is a visual Java bean. It allows visualization of XML documents and editing. It enables the display of XML and XSL formatted files with color syntax highlighting when modifying an XML document with an editing application. This helps view and edit the files. It can be easily integrated with DOM Builder Bean, and allows for pre or post parsing, and validation against a specified DTD. See "Using XMLSourceView Bean (oracle.xml.srcviewer API)" on page 20-16.

XMLTransformPanel Bean

A visual Java bean that applies XSL transformations on XML documents and shows the results. It allows editing of XML and XSL input files. See "Using XMLTransformPanel() Bean (oracle.xml.transviewer API)" on page 20-20.

Using the XML Transviewer Beans

Guidelines for using the XML Transviewer Beans are described in the following sections:

- Using DOMBuilder Bean (Async API)
- Using XSL Transformer Bean (oracle.xml.async API)
- Using Treeviewer Bean (XMLTreeView() API)
- Using XMLSourceView Bean (oracle.xml.srcviewer API)
- Using XMLTransformPanel() Bean (oracle.xml.transviewer API)

> **See Also:** *Oracle8i XML Reference* and in this manual, Chapter 3, "Oracle XML Components and General FAQs".

Using DOMBuilder Bean (Async API)

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

The parsing is done in a separate thread and DOMBuilderListener interface must be used for notification when the tree is built.

Used for Asynchronous Parsing in the Background

DOM Builder Bean encapsulates XML Parser for Java with a bean interface. It extends its functionality to permit asynchronous parsing. By registering a listener, a Java application can parse documents and return control return to the caller.

Asynchronous parsing in a background thread can be used interactively in visual applications. For example, when parsing a large file with the normal parser, the user interface can freeze till the parsing has completed. This can be avoided with the DOMBuilder Bean. After calling the DOMBuilder Bean parse method, the application can receive control back immediately and display “Parsing, please wait”. If a “Cancel” button is included you can also cancel the operation. The application can continue when `domBuilderOver()` method is called by DOMBuilder bean when background parsing task has completed.

DOMBuilder Bean Parses Many Files Fast

When parsing a large number of files, DOMBuilder Bean can save you much time. Up to 40% faster times have been recorded when compared to parsing the files one by one.

DOMBuilder Bean Usage

Figure 20–1 illustrates DOMBuilder Bean usage.

1. The XML document to be parsed is input as a file, string buffer, or URL.
2. This inputs `DOMBuilder.addDOMBuilderListener(DOMBuilderListener)`method. This adds DOMBuilderListener.
3. The DOMBuilder.parser() method parses the XML document.
4. Optionally, the parsed result undergoes further processing. See Table 20–1 for a list of available methods to apply.

5. DOMBuilderListener API is called using DOMBuilderOver() method. This is called when it received an async call from an application. This interface must be implemented to receive notifications about events during asynchronous parsing. The class implementing this interface must be added to the DOMBuilder using addDOMBuilderListener method.

   Available DOMBuilderListener methods are:

   - domBuilderError(DOMBuilderEvent). This method is called when parse error occur.
   - domBuilderOver(DOMBuilderEvent). This method is called when the parse is complete
   - domBuilderStarted(DOMBuilderEvent). This method is called when parsing starts

6. DOMBuilder.getDocument() fetches the resulting DOM document and outputs the DOM document.

Figure 20–1 DOMBuilder Bean Usage

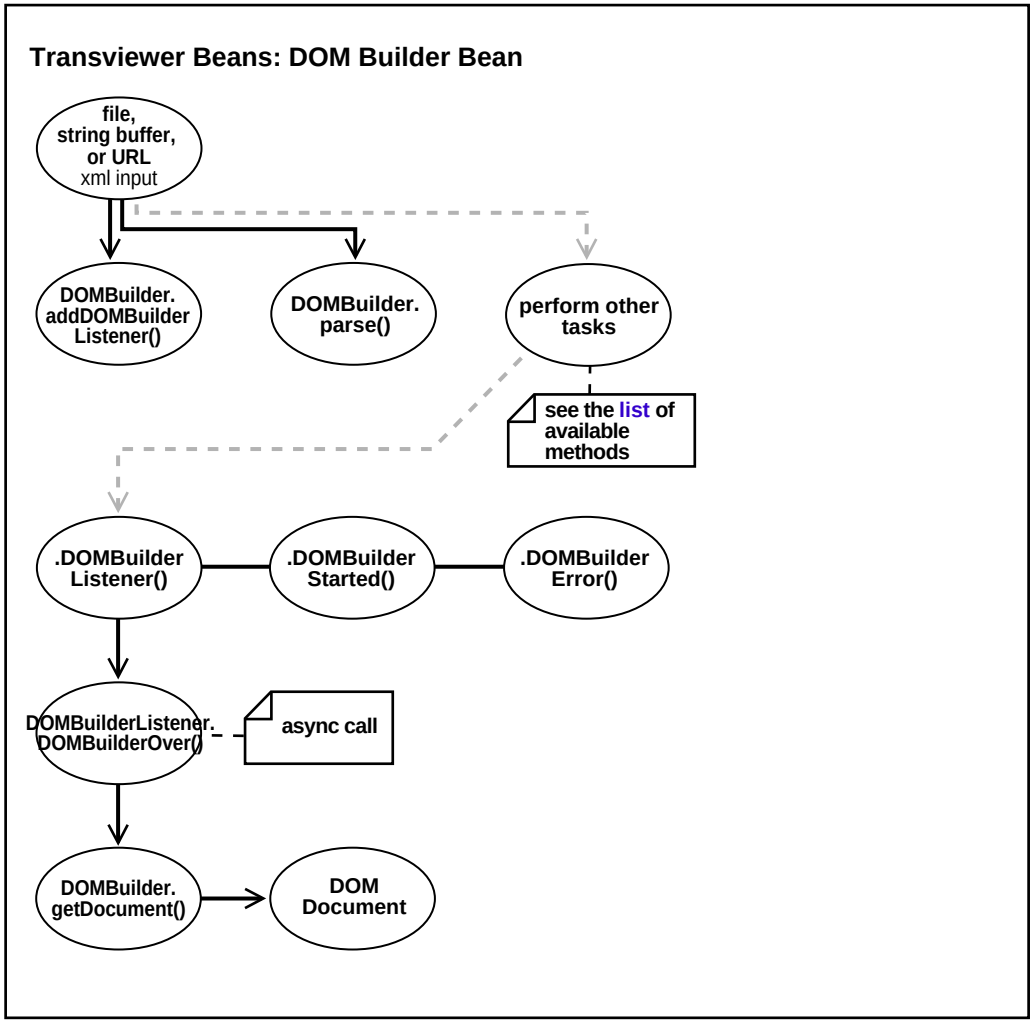


Table 20–1 DOMBuilder Bean (Async API): Methods

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)	Remove DOMBuilderErrorListener
removeDOMBuilderListener(DOMBuilderListener)	Remove DOMBuilderListener
run()	This method runs in a thread

Table 20–1 DOMBuilder Bean (Async API): 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)	Set 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.
	Set the node factory.
setPreserveWhitespace(boolean)	Set the white space preserving mode
setValidationMode(boolean)	Set the validation mode
showWarnings(boolean)	Switch to determine whether to print warnings

Using XSL Transformer Bean (oracle.xml.async API)

The XSL Transformer 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, XSL Transformer Bean enables an application or user to view the results of transformations immediately.

This bean can also be used as the basis of a server-side application or servlet to render an XML document, such as an XML representation of a query result, into HTML for display in a browser.

The XSL Transformer Bean encapsulates the Java XML Parser XSL-T processing engine with a bean interface and extends its functionality to permit asynchronous transformation.

By registering a listener, your Java application can transform large and successive documents by having the control returned immediately to the caller.

Many Files to Transform? Use XSL Transformer Bean

XSL transformations can be time consuming. Use XSL Transformer bean in applications that transform large number of files. It can transform multiple files concurrently.

Need a responsive User Interface? Use XSL Transformer Bean

XSL Transformer 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 Underlying Data Changes

This scenario illustrates one way of applying XSL Transformer Bean.

1. Create a SQL query. Store the selected XML data in a CLOB table.

2. Using the XSL Transfomer Bean, create an XSL stylesheet and interactively apply this to the XML data until you are satisfied by the data presentation. This can be HTML produced by the XSL transformation.
3. Now that you have the desired SQL (data selection) and XSL (data presentation), create a trigger on the table or view used by your SQL query. The trigger can execute a stored procedure. The stored procedure, can for example, do the following:
   - Run the query
   - Apply the stylesheet
   - Store the resulting HTML in a CLOB table.
4. This process can repeat whenever the source data table is updated.

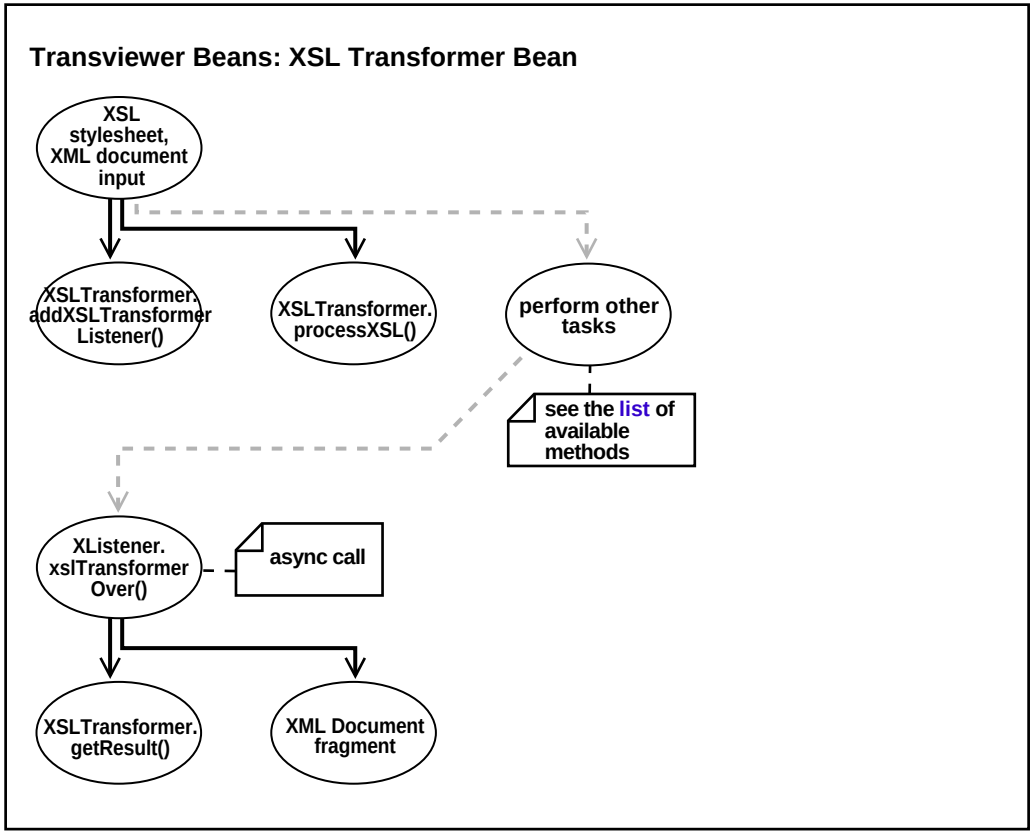
   The HTML stored in the CLOB table always mirrors the last data stored in the tables being queried. A JSP (Java Server Page) can display the HTML.

   In this scenario, multiple end users do not produce multiple data queries that contribute to bigger loads to the database. The HTML is regenerated only when the underlying data changes and only then.

XSL Transformer Bean Usage

Figure 20–2 illustrates the XSL Transformer Bean usage. For examples of implementing this bean, see "Transviewer Bean Example 1: AsyncTransformSample.java".

Figure 20–2 XSL Transformer Bean Usage



1. An XSL stylesheet and XML document input the XSLTransformer using method: `XSLTransfomer.addXSLTransformerListener(XSLTransformerListener).` 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 20–2 lists available XSLTransformer Bean methods.
4. When the transformation is complete, an asynchronous call is made and the `XSLTransformerListener.xslTransformerOver()` method is called.

This interface must be implemented in order to receive notifications about events during the asynchronous transformation. The class implementing this interface must be added to the XSLTransformer event queue using `addXSLTransformerListener` method.

5. XSLTransformer.getResult() method returns the XML document fragment for the resulting document.
6. It ouputs the XML document fragment.

Table 20–2 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 (XMLTreeView() API)

The Treeviewer Bean shows 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.

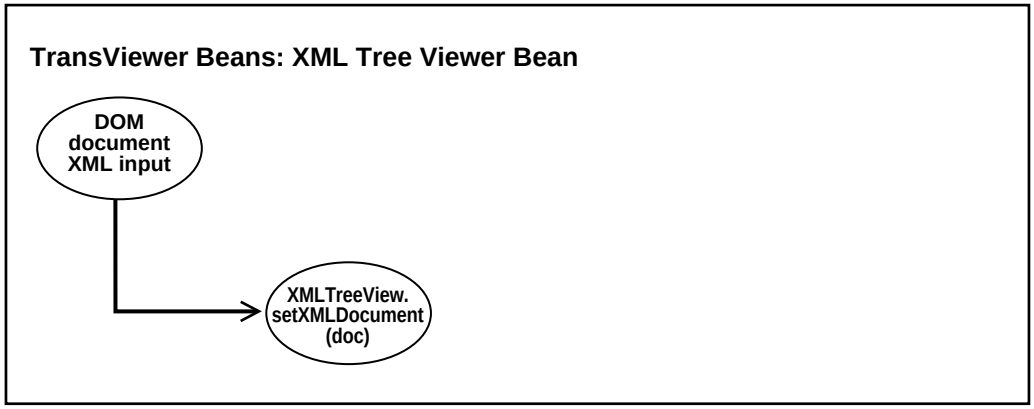
Figure 20–3 illustrates the XML Tree Viewer Bean usage.

A DOM XML document is input to the XMLTreeView.setXMLDocument(doc) method. This associates the XML Tree Viewer with the XML document.

Available 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 20–3 XML Tree Viewer Bean Usage



Using XMLSourceView Bean (oracle.xml.srcviewer API)

XML Sourceview Bean is a visual Java bean that shows an XML document. It improves the viewing of XML and XSL files by color-highlighting the XML / XSL syntax. It enables an Edit mode.

XMLSourceView Bean easily integrates with DOM Builder Bean. It allows for pre or post parsing visualization and validation against a specified DTD.

XMLSourceView Bean recognizes the following XML token types:

- Tag
- Attribute Name
- Attribute Value
- Comment
- CDATA
- PCDATA
- PI Data
- PI Name
- NOTATION Symbol

Each token type has a foreground color and font. The default color / font settings can be changed by the user. This takes as input an org.w3c.dom.Document object.

XMLSourceView Bean Usage

Figure 20–4 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".

Table 20–3 lists XMLSourceView methods.

Figure 20–4 XMLSourceView Bean Usage

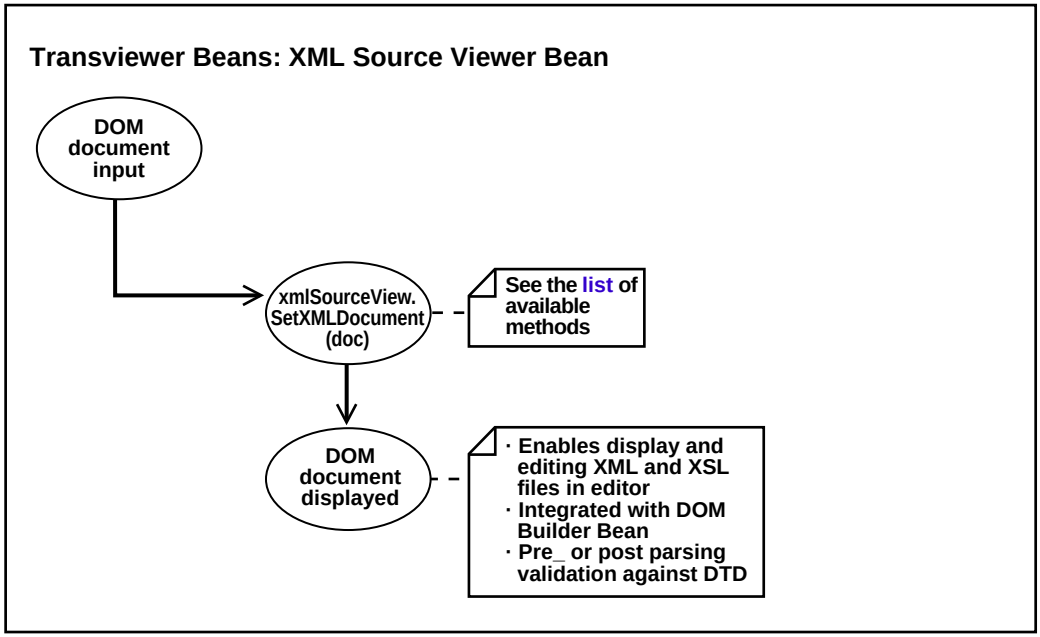


Table 20–3 XMLSourceView Bean Methods

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.

Table 20–3 XMLSourceView Bean Methods (Cont.)

Method	Description
getCommentDataForeground()	Returns the Comment Data foreground color.
getEditedText()	Returns the edited text.
getJTextPane()	Returns the viewer JTextPane component.
getMinimumSize()	Returns the XMLSourceView minimal size.
getNodeAtOffset(int)	Returns the XML node at a given offset.
getPCDATAFont()	Returns the PCDATA font.
getPCDATAForeground()	Returns the PCDATA foreground color.
getPIDataFont()	Returns the PI Data font.
getPIDataForeground()	Returns the PI Data foreground color.
getPINameFont()	Returns the PI Name font.
getPINameForeground()	Returns the PI Data foreground color.
getSymbolFont()	Returns the NOTATION Symbol font.
getSymbolForeground()	Returns the NOTATION Symbol foreground color.
getTagFont()	Returns the Tag font.
getTagForeground()	Returns the Tag foreground color.
getText()	Returns the XML document as a String.
isEditable()	Returns boolean to indicate whether this object is editable.
selectNodeAt(int)	Moves the cursor to XML Node at offset i.
setAttributeNameFont(Font)	Sets the Attribute Name font.
setAttributeNameForeground(Color)	Sets the Attribute Name foreground color.
setAttributeValueFont(Font)	Sets the Attribute Value font.
setAttributeValueForeground(Color)	Sets the Attribute Value foreground color.
setBackground(Color)	Sets the background color.
setCDATAFont(Font)	Sets the CDATA font.
setCDATAForeground(Color)	Sets the CDATA foreground color.
setCommentDataFont(Font)	Sets the Comment font.
setCommentDataForeground(Color)	Sets the Comment foreground color.

Table 20–3 XMLSourceView Bean Methods (Cont.)

Method	Description
setEditable(boolean)	Sets the specified boolean to indicate whether this object should be editable.
setPCDATAFont(Font)	Sets the PCDATA font.
setPCDATAForeground(Color)	Sets the PCDATA foreground color.
setPIDataFont(Font)	Sets the PI Data font.
setPIDataForeground(Color)	Sets the PI Data foreground color.
setPINameFont(Font)	Sets the PI Name font.
setPINameForeground(Color)	Sets the PI Name foreground color.
setSelectedNode(Node)	Sets the cursor position at the selected XML node.
setSymbolFont(Font)	Sets the NOTATION Symbol font.
setSymbolForeground(Color)	Sets the NOTATION Symbol foreground color.
setTagFont(Font)	Sets the Tag font.
setTagForeground(Color)	Sets the Tag foreground color.
setXMLDocument(Document)	Associates the XMLviewer with a XML document.

Using XMLTransformPanel() Bean (oracle.xml.transviewer API)

XMLTransformPanel visual bean applies XSL transformations to XML documents. It visualizes the result and allows editing of input XML and XSL documents and files.

This bean required no programmatic input. It is a component that interacts directly with you and is not customizable.

XMLTransformPanel Bean Features

XMLTransformPanel Bean also has the following features:

- Imports and exports XML and XSL files from the file system, and XML, XSL, and HTML files from Oracle8i. With Oracle8*i*, XSL Transformer Bean uses two-column CLOB tables. The first column stores the data name (file name) and the second stores the data text (file's data) in a CLOB. The bean lists all CLOB tables in your schema. When you click on a table, the bean lists its file names. You can also create or delete tables, retrieve or add files to the tables. This can be useful for organizing your information.

  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 Underlying Data Changes".

- Supports multiple database connections
- Creates XML from database result sets. This feature allows you to submit any SQL query to the database that you are currently connected. The bean converts the result set into XML and automatically loads this XML data into the beans XML buffer for further processing.
- Edits XML and XSL data loaded into the bean.
- Applies XSL transformations to XML buffers and show the results. With the Bean, you can also export results to the file system or a CLOB in the database.

Transviewer Bean

This could be confusing to you, because *all* the beans in this chapter (and in XDK for Java) are referred to as "Transviewer Beans", but the *specific* "Transviewer Bean" is an

application that uses XMLTransformPanel Bean, in other words, it uses the `oracle.xml.transviewer()` API.

Command Line Usage

Transviewer Bean can be used from the 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 Oracle8*i*

DBAccess() Class

This class 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. The CLOB tables are automatically maintained by the Transviewer Bean. The CLOB tables maintained by this class can be later used by the Transviewer Bean.

The class does the following tasks:

- Creates and deletes CLOB tables
- Lists a CLOB table contents
- Adds, replaces, or deletes text documents in the CLOB tables

> **See:** "Transviewer Bean Example 3: XMLTransformPanelSample.java" for an example of how to use XMLTransformPanel.

Running the Transviewer Bean Samples Supplied

The XDK for Java Transviewer Bean sample/ directory contains sample Transviewer Bean applications to illustrate how to use Oracle Transviewer Beans. Oracle Transviewer beans toolset contains DOMBuilder, XMLSourceView, XMLTreeView, XSLTransformer, and XMLTransformPanel beans.

Table 20–4 lists the sample files in sample/.

Table 20–4 Transviewer Bean Sample Files in sample/

File Name	Description
booklist.xml	Sample XML file used by Example 1, 2, or 3
doc.xml	Sample XML file used by Example 1, 2, or 3
doc.html	Sample HTML file used by Examples 1, 2, or 3
doc.xsl	Sample input XSL file used by Examples 1, 2, or 3. doc.xsl is used by XSLTransformer.
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 XSL-T 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.
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.

Installing the Transviewer Bean Samples

The Transviewer Beans require as minimum JDK 1.1.6 and can be used with any version of JDK 1.2.

1. Download and install the following components that are used by the Transviewer beans:

   - Oracle JDBC Driver for thin client:

     If you work with JDK 1.2 and up, use the 1.2 version of the JDBC driver from file classes12.zip. For JDK 1.1.6 and up, use file classes111.zip.

     Ensure that the version of the JDBC driver is the same or higher than the Oracle database you want to access. The database tested with the beans was Oracle 8.1.5

   - Oracle XML-SQL Utility (jar file oraclexmlsql.jar)

     After installing the components, include classes111.zip and oraclexmlsql.jar into your classpath.

2. The beans and the samples use swing 1.1. If you use jdk1.2, go to step 3. If you use jdk1.1, you will need to download Swing 1.1 from Sun. After downloading Swing, add swingall.jar to your classpath.

3. Change JDKPATH in `Makefile` to point to your JDK path.

   In addition, on Windows NT, change the file separator as stated in the `Makefile`.

4. Run "**`make`**" to generate .class files.

5. Run the sample programs using commands:

   - **gmake** sample1
   - **gmake** sample2
   - **gmake** sample3

6. Visualize the results in .log files using the ViewSample.

7. Use the XSL-T document from './tohtml.xsl' to transform the XML document from './booklist.xml'.

A few .xml files are provided as test cases.

Note: sample1 runs the XMLTransViewer program so that you can import and export XML files from Oracle8*i*, keep your XSL transformation files in Oracle8*i*, and apply stylesheets to XML interactively.

Setting Up Your Environment to Run the Samples

To use the database connectivity feature in this program, you must know the following:

- Network name of the computer where Oracle8*i or i*AS 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 Oracle8i 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 styelsheets
- 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 ();
  }
}
```

Part VII

XDK for C

Part VII describes how to access and use the XML Development Kit (XML) for C.

It contains the following chapters:

21

Using XML Parser for C

This chapter contains the following sections:

Accessing XML Parser for C

The XML Parser for C is provided with Oracle8*i* and is also available for download from the OTN site: http: / / technet.oracle.com / tech / xml

It is located in $ORACLE_HOME / xdk / c / parser.

XML Parser for C Features

`readme.html` in the root directory of the software archive contains release specific information including bug fixes and API additions.

XML Parser for C will check if an XML document is well-formed, and optionally validate it against a DTD. The parser constructs an object tree which can be accessed through a DOM interface or operate serially via a SAX interface.

You can post questions, comments, or bug reports to the XML Discussion Forum at http: / / technet.oracle.com.

Specifications

See Appendix D, "XDK for C: Specifications and Cheat Sheets" for a brief list of XML Parser for C methods and specifications.

See Also:

- The doc directory in your install area
- *Oracle8i XML Reference*
- On OTN under http: / / technet.oracle.com / tech / xml / parser_ cpp / index.htm

Memory Allocation

The memory callback functions `memcb` may be used if you wish to use your own memory allocation. If they are used, all of the functions should be specified.

The memory allocated for parameters passed to the SAX callbacks or for nodes and data stored with the DOM parse tree will not be freed until one of the following is done:

- `xmlparse()` or `xmlparsebuf()` is called to parse another file or buffer.
- `xmlclean()` is called.
- `xmlterm()` is called.

Thread Safety

If threads are forked off somewhere in the midst of the init-parse-term sequence of calls, you will get unpredictable behavior and results.

Data Types Index

Table 21–1 lists the datatypes used in XML Parser for C.

Table 21–1 Datatypes Used in XML Parser for C

DataType	Description
oratext	String pointer
xmlctx	Master XML context
xmlmemcb	Memory callback structure (optional)
xmlsaxcb	SAX callback structure (SAX only)
ub4	32-bit (or larger) unsigned integer
uword	Native unsigned integer

Error Message Files

Error message files are provided in the mesg/ subdirectory. The messages files also exist in the `$ORACLE_HOME/oracore/mesg` directory. You may set the environment variable `ORA_XML_MESG` to point to the absolute path of the `mesg/` subdirectory although this not required.

XML Parser for C Usage

Figure 21–1 illustrates the XML Parser for C functionality as follows:

1. `XMLinit()` function initializes the parsing process.
2. The parsed item can be an XML document (file) or string buffer. If the input is an XML document or file, it is parsed using the `xmlparser()` function. If the input is a string buffer, it is parsed using the `xmlparserbuf()` function.
3. DOM or SAX API:

   DOM: If you are using the DOM interface, include the following steps:

   - The `xmlparse()` or `xmlparseBuffer()` function calls `.getDocumentElement()`. If no other DOM functions are being applied, you can invoke `xmlterm()`.
   - This optionally calls other DOM functions if required. These are typically Node or print functions. It outputs the DOM document.
   - If complete, the process invokes `xmlterm()`
   - You can optionally first invoke `xmlclean()` to clean up any data structures created during the parse process. You would then call `xmlterm()`

   SAX: If you are using the SAX interface, include the following steps:

   - Process the results of the parser from `xmlparse()` or `xmlparseBuf()` using callback functions.
   - Register the callback functions.
4. Optionally, use `xmlclean()` to clean up the memory and structures used during a parse, and go to Step 5. or return to Step 2.
5. Terminate the parsing process with `xmlterm()`

Parser Calling Sequence

The sequence of calls to the parser can be any of the following:

- `xmlinit() - xmlparse()` or

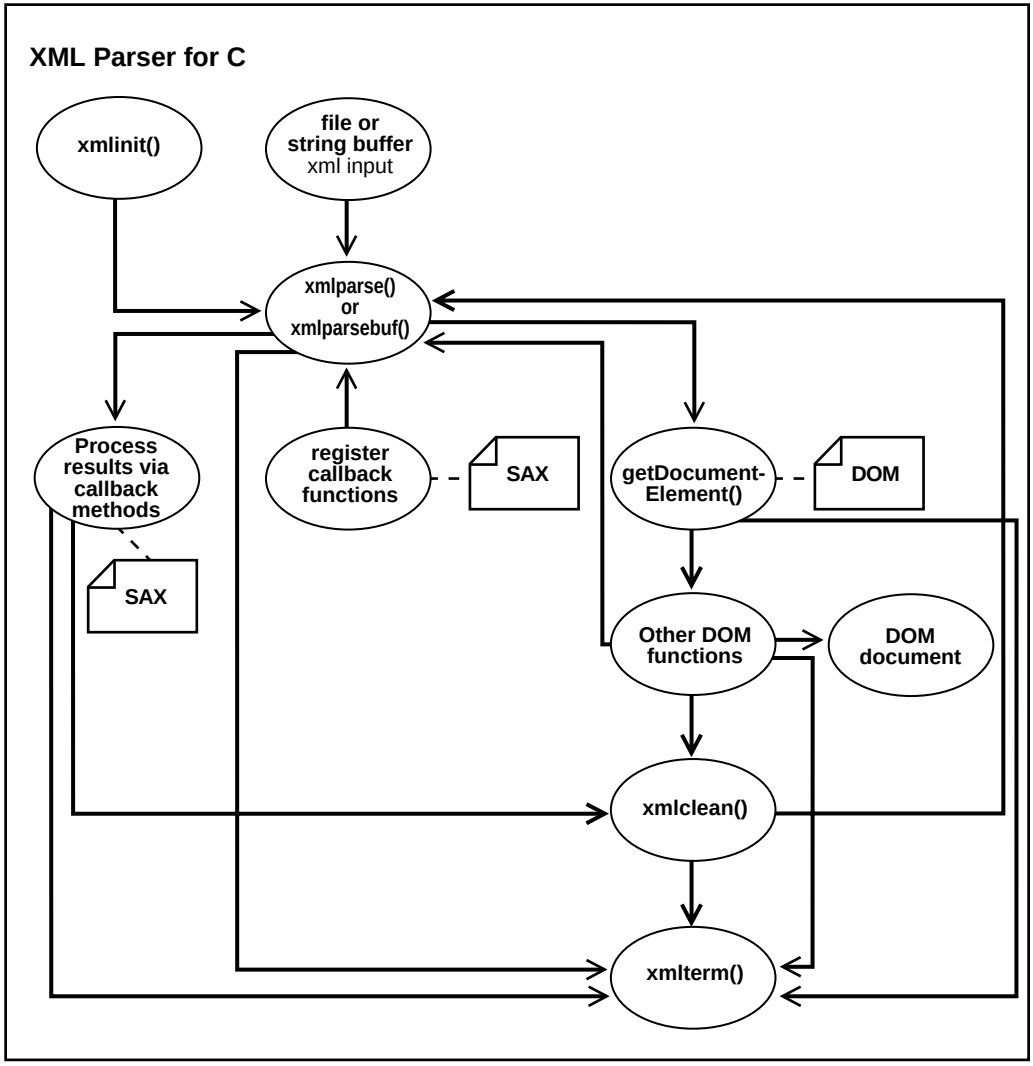
  `xmlparsebuf() - xmlterm()`
- `xmlinit() - xmlparse()` or

  `xmlparsebuf() - xmlclean() - xmlparse()` or

```
xmlparsebuf() - xmlclean() - ... - xmlterm()
```

- `xmlinit() - xmlparse()` or
  `xmlparsebuf() - xmlparse()` or
  `xmlparsebuf() -... - xmlterm()`

Figure 21–1 XML Parser for C Usage



XML Parser for C XSLT (DOM Interface) Usage

Figure 21–2 shows the overall XML Parser for C XSLT functionality.

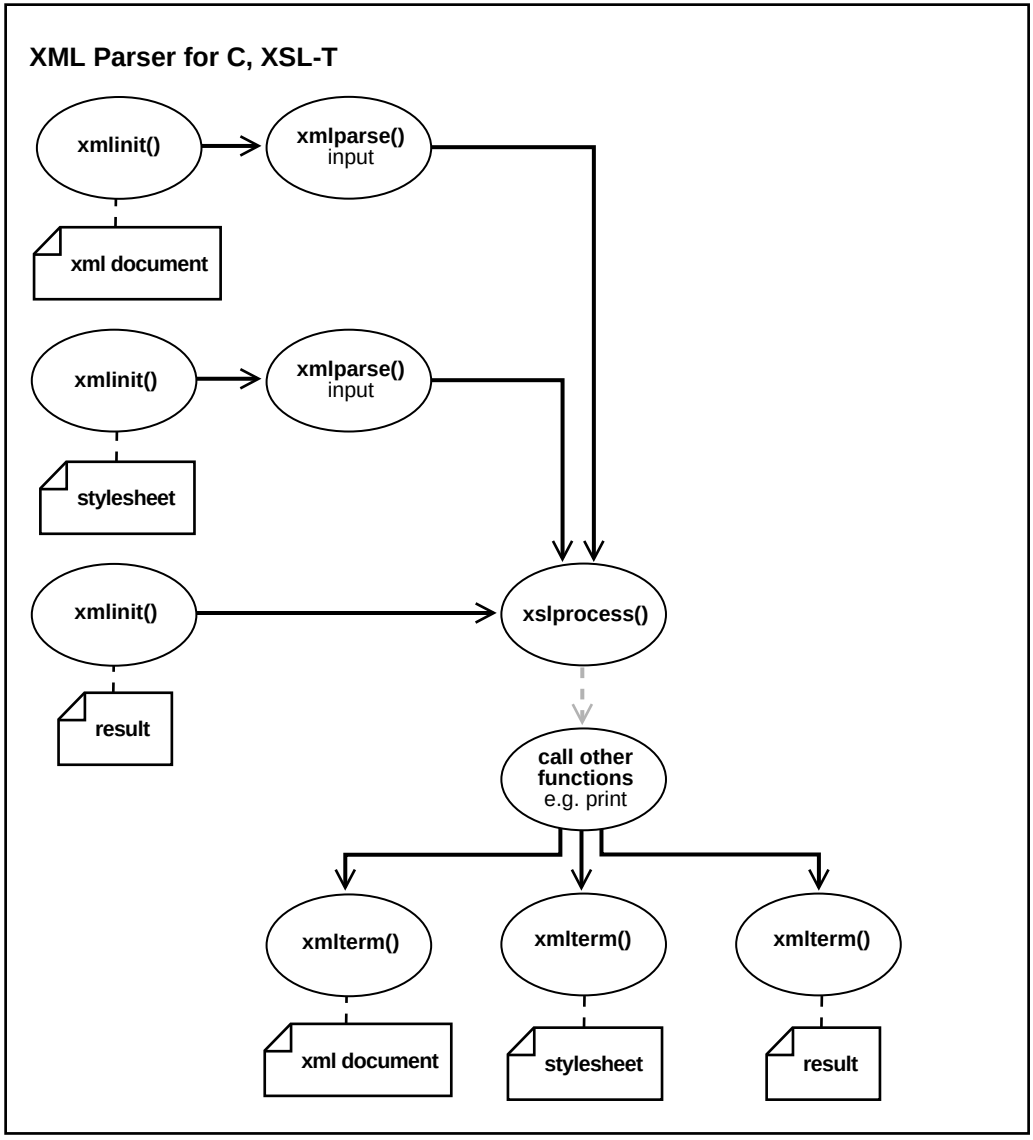
1. There are two inputs to `xmlparse()`:
   - The Stylesheet to be applied to the XML document
   - XML document

   The output of `xmlparse()`, the parsed stylesheet and parsed XML document, are sent to the `xslprocess()` function for processing.
2. `xmlinit()` initializes the XSLT processing. `xmlinit()` initializes the `xslprocess()` result
3. `xslprocess()`optionally calls other functions, such as print functions. You can see the list of available functions either on OTN or in the *Oracle8i XML Reference*.
4. The resultant document (XML, HTML, VML, and so on) is typically sent to an application for further processing.
5. The application terminates the XSLT process by declaring `xmlterm()`, for the XML document, stylesheet, and final result.

XML Parser for C's XSLT functionality is illustrated with the following examples:

- XML Parser for C Example 16: C — XSLSample.c on page 21-53
- XML Parser for C Example 17: C — XSLSample.std on page 21-55

Figure 21–2 XML Parser for C: XSL-T (DOM Interface) Usage



Default Behavior

The following is the XML Parser for C default behavior:

- Character set encoding is UTF-8. If all your documents are ASCII, you are encouraged to set the encoding to US-ASCII for better performance.
- Messages are printed to stderr unless msghdlr is given.
- A parse tree which can be accessed by DOM APIs is built unless saxcb is set to use the SAX callback APIs. Note that any of the SAX callback functions can be set to NULL if not needed.
- The default behavior for the parser is to check that the input is well-formed but not to check whether it is valid. The flag XML_FLAG_VALIDATE can be set to validate the input. The default behavior for whitespace processing is to be fully conformant to the XML 1.0 spec, that is, all whitespace is reported back to the application but it is indicated which whitespace is ignorable. However, some applications may prefer to set the XML_FLAG_DISCARD_WHITESPACE which will discard all whitespace between an end-element tag and the following start-element tag.

Note: It is recommended that you set the default encoding explicitly if using only single byte character sets (such as US-ASCII or any of the ISO-8859 character sets) for performance up to 25% faster than with multibyte character sets, such as UTF-8.

DOM and SAX APIs

Oracle XML parser for C checks if an XML document is well-formed, and optionally validates it against a DTD. The parser constructs an object tree which can be accessed via one of the following interfaces:

- DOM interface
- Serially via a SAX interface

These two XML APIs:

- DOM: Tree-based APIs. A tree-based API compiles an XML document into an internal tree structure, then allows an application to navigate that tree using the Document Object Model (DOM), a standard tree-based API for XML and HTML documents.
- SAX: Event-based APIs. An event-based API, on the other hand, reports parsing events (such as the start and end of elements) directly to the application through callbacks, and does not usually build an internal tree. The application implements handlers to deal with the different events, much like handling events in a graphical user interface.

Tree-based APIs are useful for a wide range of applications, but they often put a great strain on system resources, especially if the document is large (under very controlled circumstances, it is possible to construct the tree in a lazy fashion to avoid some of this problem). Furthermore, some applications need to build their own, different data trees, and it is very inefficient to build a tree of parse nodes, only to map it onto a new tree.

In both of these cases, an event-based API provides a simpler, lower-level access to an XML document: you can parse documents much larger than your available system memory, and you can construct your own data structures using your callback event handlers.

Using the SAX API

To use SAX, an `xmlsaxcb` structure is initialized with function pointers and passed to the `xmlinit()` call. A pointer to a user-defined context structure can also be included. That context pointer will be passed to each SAX function.

SAX Callback Structure

The SAX callback structure:

```
typedef struct
```

```
{
 sword (*startDocument)(void *ctx);
 sword (*endDocument)(void *ctx);
 sword (*startElement)(void *ctx, const oratext *name,
            const struct xmlarray *attrs);
 sword (*endElement)(void *ctx, const oratext *name);
 sword (*characters)(void *ctx, const oratext *ch, size_t len);
 sword (*ignorableWhitespace)(void *ctx, const oratext *ch, size_t len);
 sword (*processingInstruction)(void *ctx, const oratext *target,
            const oratext *data);
 sword (*notationDecl)(void *ctx, const oratext *name,
            const oratext *publicId, const oratext *systemId);
 sword (*unparsedEntityDecl)(void *ctx, const oratext *name,
            const oratext *publicId,
            const oratext *systemId, const oratext *notationName);
 sword (*nsStartElement)(void *ctx, const oratext *qname,
            const oratext *local, const oratext *nsp,
            const struct xmlnodes *attrs);
} xmlsaxcb;
```

Using the DOM API

See "XML Parser for C Example 7: C — DOMSample.std" on page 21-19.

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 21–2 lists the command line options.

Table 21–2 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 21–3 lists the sample files in sample/ directory.

Table 21–3 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 ..sample/ and read the README file. This will explain how to build the sample programs according to your platform.

Sample Programs

Table 21–4 lists the programs built by the sample files in sample/

Table 21–4 XML Parser for C: Sample Built Programs in sample/

Built Program	Description
DOMSample	A sample application using DOM APIs (shows an outline of Cleopatra, i.e. the XML elements ACT and SCENE).
SAXSample [word]	A sample application using SAX APIs. Given a word, shows all lines in the play Cleopatra containing that word. If no word is specified, 'death' is used.
DOMNamespace	Same as SAXNamespace except using DOM interface.
SAXNamespace	A sample application using Namespace extensions to SAX API; prints out all elements and attributes of NSExample.xml along with full namespace information.
FullDOM	Sample usage of full DOM interface. Exercises all the calls, but does nothing too exciting.
XSLSample <xmlfile> <xsl ss>	Sample usage of XSL processor. It takes two filenames as input, the XML file and XSL stylesheet

XML Parser for C Example 1: XML — class.xml

`class.xml` is an XML file that inputs XSLSample.c.

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

```
<Dept>Math</Dept>
<Instructor>
<Name>Jim Green</Name>
</Instructor>
<Student>
<Name>Jack</Name>
<Name>Mary</Name>
<Name>Paul</Name>
</Student>
</course>
```

XML Parser for C Example 2: XML — cleo.xml

This XML example inputs `DOMSample.c` and `SAXSample.c`.

```
<?xml version="1.0"?>
<!DOCTYPE PLAY [
    <!ELEMENT PLAY     (TITLE, PERSONAE, SCNDESCR, PLAYSUBT, INDUCT?,
 PROLOGUE?, ACT+, EPILOGUE?)>
    <!ELEMENT TITLE    (#PCDATA)>
    <!ELEMENT FM       (P+)>
    <!ELEMENT P        (#PCDATA)>
    <!ELEMENT PERSONAE (TITLE, (PERSONA | PGROUP)+)>
    <!ELEMENT PGROUP   (PERSONA+, GRPDESCR)>
    <!ELEMENT PERSONA  (#PCDATA)>
    <!ELEMENT GRPDESCR (#PCDATA)>
    <!ELEMENT SCNDESCR (#PCDATA)>
    <!ELEMENT PLAYSUBT (#PCDATA)>
    <!ELEMENT INDUCT   (TITLE, SUBTITLE*, (SCENE+|(SPEECH|STAGEDIR|SUBHEAD)+))>
    <!ELEMENT ACT      (TITLE, SUBTITLE*, PROLOGUE?, SCENE+, EPILOGUE?)>
    <!ELEMENT SCENE    (TITLE, SUBTITLE*, (SPEECH | STAGEDIR | SUBHEAD)+)>
    <!ELEMENT PROLOGUE (TITLE, SUBTITLE*, (STAGEDIR | SPEECH)+)>
    <!ELEMENT EPILOGUE (TITLE, SUBTITLE*, (STAGEDIR | SPEECH)+)>
    <!ELEMENT SPEECH   (SPEAKER+, (LINE | STAGEDIR | SUBHEAD)+)>
    <!ELEMENT SPEAKER  (#PCDATA)>
    <!ELEMENT LINE     (#PCDATA | STAGEDIR)*>
    <!ELEMENT STAGEDIR (#PCDATA)>
    <!ELEMENT SUBTITLE (#PCDATA)>
    <!ELEMENT SUBHEAD  (#PCDATA)>
]>

<PLAY>
<TITLE>The Tragedy of Antony and Cleopatra</TITLE>
```

```
<PERSONAE>
<TITLE>Dramatis Personae</TITLE>


<PGROUP>
<PERSONA>MARK ANTONY</PERSONA>
<PERSONA>OCTAVIUS CAESAR</PERSONA>
<PERSONA>M. AEMILIUS LEPIDUS</PERSONA>
<GRPDESCR>triumvirs.</GRPDESCR>
</PGROUP>

<PERSONA>SEXTUS POMPEIUS</PERSONA>

<PGROUP>
<PERSONA>DOMITIUS ENOBARBUS</PERSONA>
<PERSONA>VENTIDIUS</PERSONA>
<PERSONA>EROS</PERSONA>
<PERSONA>SCARUS</PERSONA>
<PERSONA>DERCETAS</PERSONA>
<PERSONA>DEMETRIUS</PERSONA>
<PERSONA>PHILO</PERSONA>
<GRPDESCR>friends to Antony.</GRPDESCR>
</PGROUP>


<PGROUP>
<PERSONA>MECAENAS</PERSONA>
<PERSONA>AGRIPPA</PERSONA>
<PERSONA>DOLABELLA</PERSONA>
<PERSONA>PROCULEIUS</PERSONA>
<PERSONA>THYREUS</PERSONA>
<PERSONA>GALLUS</PERSONA>
<PERSONA>MENAS</PERSONA>
<GRPDESCR>friends to Caesar.</GRPDESCR>
</PGROUP>

...
...

<SCNDESCR>SCENE  In several parts of the Roman empire.</SCNDESCR>

<PLAYSUBT>ANTONY AND CLEOPATRA</PLAYSUBT>

<ACT><TITLE>ACT I</TITLE>
```

```
<SCENE><TITLE>SCENE I.  Alexandria. A room in CLEOPATRA's palace.</TITLE>
<STAGEDIR>Enter DEMETRIUS and PHILO</STAGEDIR>

<SPEECH>
<SPEAKER>PHILO</SPEAKER>
<LINE>Nay, but this dotage of our general's</LINE>
<LINE>O'erflows the measure: those his goodly eyes,</LINE>
<LINE>That o'er the files and musters of the war</LINE>
<LINE>Have glow'd like plated Mars, now bend, now turn,</LINE>
<LINE>The office and devotion of their view</LINE>
<LINE>Upon a tawny front: his captain's heart,</LINE>
<LINE>Which in the scuffles of great fights hath burst</LINE>
<LINE>The buckles on his breast, reneges all temper,</LINE>
<LINE>And is become the bellows and the fan</LINE>
<LINE>To cool a gipsy's lust.</LINE>
<STAGEDIR>Flourish. Enter ANTONY, CLEOPATRA, her Ladies,
the Train, with Eunuchs fanning her</STAGEDIR>
<LINE>Look, where they come:</LINE>
<LINE>Take but good note, and you shall see in him.</LINE>
<LINE>The triple pillar of the world transform'd</LINE>
<LINE>Into a strumpet's fool: behold and see.</LINE>
</SPEECH>

<SPEECH>
<SPEAKER>CLEOPATRA</SPEAKER>
<LINE>If it be love indeed, tell me how much.</LINE>
</SPEECH>

<SPEECH>
<SPEAKER>MARK ANTONY</SPEAKER>
<LINE>There's beggary in the love that can be reckon'd.</LINE>
</SPEECH>

<SPEECH>
<SPEAKER>CLEOPATRA</SPEAKER>
<LINE>I'll set a bourn how far to be beloved.</LINE>
</SPEECH>

<SPEECH>
<SPEAKER>MARK ANTONY</SPEAKER>
<LINE>Then must thou needs find out new heaven, new earth.</LINE>
</SPEECH>
...
...
...
```

```
<SPEAKER>DOLABELLA</SPEAKER>
<LINE>Here, on her breast,</LINE>
<LINE>There is a vent of blood and something blown:</LINE>
<LINE>The like is on her arm.</LINE>
</SPEECH>

<SPEECH>
<SPEAKER>First Guard</SPEAKER>
<LINE>This is an aspic's trail: and these fig-leaves</LINE>
<LINE>Have slime upon them, such as the aspic leaves</LINE>
<LINE>Upon the caves of Nile.</LINE>
</SPEECH>

<SPEECH>
<SPEAKER>OCTAVIUS CAESAR</SPEAKER>
<LINE>Most probable</LINE>
<LINE>That so she died; for her physician tells me</LINE>
<LINE>She hath pursued conclusions infinite</LINE>
<LINE>Of easy ways to die. Take up her bed;</LINE>
<LINE>And bear her women from the monument:</LINE>
<LINE>She shall be buried by her Antony:</LINE>
<LINE>No grave upon the earth shall clip in it</LINE>
<LINE>A pair so famous. High events as these</LINE>
<LINE>Strike those that make them; and their story is</LINE>
<LINE>No less in pity than his glory which</LINE>
<LINE>Brought them to be lamented. Our army shall</LINE>
<LINE>In solemn show attend this funeral;</LINE>
<LINE>And then to Rome. Come, Dolabella, see</LINE>
<LINE>High order in this great solemnity.</LINE>
</SPEECH>

<STAGEDIR>Exeunt</STAGEDIR>
</SCENE>
</ACT>
</PLAY>
```

XML Parser for C Example 3: XSL — iden.xsl

This example stylesheet can be used to input `XSLSample.c`.

```
<?xml version="1.0"?>
<!-- Identity transformation -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
  <xsl:template match="*|@*|comment()|processing-instruction()|text()">
      <xsl:copy>
          <xsl:apply-templates
select="*|@*|comment()|processing-instruction()|text()"/>
      </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

XML Parser for C Example 4: XML — FullDOM.xml (DTD)

This DTD example inputs `FullDOM.c`.

```
<!DOCTYPE doc [
    <!ELEMENT p (#PCDATA)>
    <!ATTLIST p xml:space (preserve|default) 'preserve'>
    <!NOTATION notation1 SYSTEM "file.txt">
    <!NOTATION notation2 PUBLIC "some notation">
    <!ELEMENT doc (p*)>
    <!ENTITY example "<p>An ampersand (&#38;#38;) may be escaped
numerically (&#38;#38;#38;) or with a general entity
(&amp;amp;).</p>">
]>
<doc xml:lang="foo">&example;</doc>
```

XML Parser for C Example 5: XML — NSExample.xml

The following example file, `NSExample.xml`, uses namespaces.

```
<!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>
```

XML Parser for C Example 6: C — DOMSample.c

This example contains the C source code for `DOMSample.c`

```
/* Copyright (c) Oracle Corporation 1999. All Rights Reserved. */
/*
   NAME
     DOMSample.c - Sample DOM usage
   DESCRIPTION
     Sample usage of C XML parser via DOM interface
*/

#include <stdio.h>

#ifndef ORATYPES
# include <oratypes.h>
#endif
#ifndef ORAXML_ORACLE
# include <oraxml.h>
#endif

#define DOCUMENT        (oratext *) "cleo.xml"

void dump(xmlctx *ctx, xmlnode *node);
void dumppart(xmlctx *ctx, xmlnode *node, boolean indent);

int main()
{
    xmlctx     *ctx;
    uword       ecode;

    puts("XML C DOM sample");
    puts("Initializing XML package...");
    if (!(ctx = xmlinit(&ecode, (const oratext *) 0,
                        (void (*)(void *, const oratext *, uword)) 0,
                        (void *) 0, (const xmlsaxcb *) 0, (void *) 0,
(const xmlmemcb *) 0, (void *) 0,
                        (const oratext *) 0)))
    {
printf("Failed to initialze XML parser, error %u\n", (unsigned) ecode);
return 1;
    }

    printf("Parsing '%s' ...\n", DOCUMENT);
    if (ecode = xmlparse(ctx, DOCUMENT, (oratext *) 0,
XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE))
```

```
    {
printf("Parse failed, error %u\n", (unsigned) ecode);
return 1;
    }

    puts("Outlining...");
    dump(ctx, getDocumentElement(ctx));

    xmlterm(ctx);

    return 0;
}

void dump(xmlctx *ctx, xmlnode *node)
{
    const oratext *name;
    void    *nodes;
    uword    i, n_nodes;

    name = getNodeName(node);
    if (!strcmp((char *) name, "ACT"))
        dumppart(ctx, node, FALSE);
    else if (!strcmp((char *) name, "SCENE"))
        dumppart(ctx, node, TRUE);
    if (hasChildNodes(node))
    {
        nodes = getChildNodes(node);
        n_nodes = numChildNodes(nodes);
        for (i = 0; i < n_nodes; i++)
            dump(ctx, getChildNode(nodes, i));
    }
}

void dumppart(xmlctx *ctx, xmlnode *node, boolean indent)
{
    void *title = getFirstChild(node);

    if (indent)
       fputs("    ", stdout);
    puts((char *) getNodeValue(getFirstChild(title)));
}

/* end of DOMSample.c */
```

XML Parser for C Example 7: C — DOMSample.std

The DOMSample.std example file shows the expected output from DOMSample.c

```
XML C DOM sample
Initializing XML package...
Parsing 'cleo.xml' ...
Outlining...
ACT I
    SCENE I.  Alexandria. A room in CLEOPATRA's palace.
    SCENE II.  The same. Another room.
    SCENE III.  The same. Another room.
    SCENE IV.  Rome. OCTAVIUS CAESAR's house.
    SCENE V.  Alexandria. CLEOPATRA's palace.
ACT II
    SCENE I.  Messina. POMPEY's house.
    SCENE II.  Rome. The house of LEPIDUS.
...
...
...
 ACT V
    SCENE I.  Alexandria. OCTAVIUS CAESAR's camp.
    SCENE II.  Alexandria. A room in the monument.
```

XML Parser for C Example 8: C — SAXSample.c

This example contains the C source code for SAXSample.c

```
/* Copyright (c) Oracle Corporation 1999. All Rights Reserved. */

/*
   NAME
     SAXSample.c - Sample SAX usage

   DESCRIPTION
     Sample usage of C XML parser via SAX interface
*/

#include <stdio.h>

#ifndef ORATYPES
# include <oratypes.h>
#endif

#ifndef ORAXML_ORACLE
```

```
# include <oraxml.h>
#endif

#define DOCUMENT"cleo.xml"
#define DEFAULT_KEYWORD"death"

char    *keyword;
size_t   keylen;
oratext *elem;
oratext speaker[80];

oratext *findsub(oratext *buf, size_t bufsiz, oratext *sub, size_t subsiz);
void     savestr(oratext *buf, oratext *s, size_t len);

/* SAX callback functions */

sword startDocument(void *ctx);
sword endDocument(void *ctx);
sword startElement(void *ctx, const oratext *name,
                   const struct xmlnodes *attrs);
sword endElement(void *ctx, const oratext *name);
sword characters(void *ctx, const oratext *ch, size_t len);

xmlsaxcb saxcb = {
    startDocument,
    endDocument,
    startElement,
    endElement,
    characters
};

int main(int argc, char **argv)
{
    xmlctx     *ctx;
    ub4         flags;
    uword       ecode;
    flags = XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE;

    puts("XML C SAX sample");
    keyword = (argc > 1) ? argv[1] : DEFAULT_KEYWORD;
    keylen = strlen(keyword);
    puts("Initializing XML package...");

    if (!(ctx = xmlinit(&ecode, (const oratext *) 0,
(void (*)(void *, const oratext *, uword)) 0,
```

```
(void *) 0, &saxcb, (void *) 0,
(const xmlmemcb *) 0, (void *) 0,
                        (const oratext *) 0)))
    {
        (void) printf("Failed to initialize XML parser, error %u\n",
                        (unsigned) ecode);
        return 1;
    }

    printf("Parsing '%s' and looking for lines containing '%s'...\n",
DOCUMENT, keyword);
    elem = (oratext *) "";
    if (ecode = xmlparse(ctx, (oratext *) DOCUMENT, (oratext *) 0, flags))
return 1;

    (void) xmlterm(ctx);/* terminate XML package */

    return 0;
}

sword startDocument(void *ctx)
{
    puts("startDocument");
    return 0;
}

sword endDocument(void *ctx)
{
    puts("endDocument");
    return 0;
}

sword startElement(void *ctx, const oratext *name,
                   const struct xmlnodes *attrs)
{
    elem = (oratext *) name;
    return 0;
}

sword endElement(void *ctx, const oratext *name)
{
    elem = (oratext *) "";
    return 0;
}
```

```
sword characters(void *ctx, const oratext *ch, size_t len)
{
    if (!strcmp((char *) elem, "SPEAKER"))
savestr(speaker, (oratext *) ch, len);
    else if (findsub((oratext *) ch, len, (oratext *) keyword, keylen))
printf("    %s: %.*s\n", speaker, len, ch);
    return 0;
}

oratext *findsub(oratext *buf, size_t bufsiz, oratext *sub, size_t subsiz)
{
    uword i;

    if (!buf || !bufsiz || (subsiz > bufsiz))
return (oratext *) 0;
    if (!sub || !subsiz)
return buf;
    for (i = 0; i < bufsiz - subsiz; i++, buf++)
    {
if (!memcmp(buf, sub, subsiz))
    return buf;
    }
    return (oratext *) 0;
}

void savestr(oratext *buf, oratext *s, size_t len)
{
    memcpy(buf, s, len);
    buf[len] = 0;
}

/* End of SAXSample.c */
```

XML Parser for C Example 9: C — SAXSample.std

`SAXSample.std` shows the expected output from `SAXSample.c`.

```
XML C SAX sample
Initializing XML package...
Parsing 'cleo.xml' and looking for lines containing 'death'...
startDocument
    MARK ANTONY: Who tells me true, though in his tale lie death,
    DOMITIUS ENOBARBUS: if they suffer our departure, death's the word.
    DOMITIUS ENOBARBUS: mettle in death, which commits some loving act upon
```

```
    MARK ANTONY: The death of Fulvia, with more urgent touches,
    MARK ANTONY: Is Fulvia's death.
    CLEOPATRA: In Fulvia's death, how mine received shall be.
    EROS: the poor third is up, till death enlarge his confine.
    SCARUS: Where death is sure. Yon ribaudred nag of Egypt,--
    EROS: Her head's declined, and death will seize her, but
    MARK ANTONY: I'll make death love me; for I will contend
    MARK ANTONY: Married to your good service, stay till death:
    MARK ANTONY: Than death and honour. Let's to supper, come,
    First Soldier: The hand of death hath raught him.
    CLEOPATRA: And bring me how he takes my death.
    MARK ANTONY: She hath betray'd me and shall die the death.
    MARK ANTONY: Than she which by her death our Caesar tells
    EROS: Of Antony's death.
    MARK ANTONY: A bridegroom in my death, and run into't
    DERCETAS: Thy death and fortunes bid thy followers fly.
    MARK ANTONY: Sufficing strokes for death.
    DIOMEDES: His death's upon him, but not dead.
    MARK ANTONY: I here importune death awhile, until
    CLEOPATRA: To rush into the secret house of death,
    CLEOPATRA: Ere death dare come to us? How do you, women?
    CLEOPATRA: And make death proud to take us. Come, away:
    OCTAVIUS CAESAR: And citizens to their dens: the death of Antony
    CLEOPATRA: What, of death too,
    CLEOPATRA: Where art thou, death?
    CLEOPATRA: The stroke of death is as a lover's pinch,
    CHARMIAN: Now boast thee, death, in thy possession lies
    OCTAVIUS CAESAR: Took her own way. The manner of their deaths?
endDocument
```

XML Parser for C Example 10: C — DOMNamespace.c

This example contains the C source code for `DOMNamespace.c`.

```
/* Copyright (c) Oracle Corporation 1999. All Rights Reserved. */

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

#ifndef ORATYPES
```

```
# include <oratypes.h>
#endif

#ifndef ORAXML_ORACLE
# include <oraxml.h>
#endif

#define DOCUMENT         "NSExample.xml"

/*---------------------------------------------------------------------------
                          FUNCTION PROTOTYPES
  ---------------------------------------------------------------------------*/
static void    DOMNSprint(xmlctx *ctx);
static void    printElements(xmlctx *ctx, xmlnode *n);
static void    printAttrs(xmlctx *ctx, xmlnode *n);

/*---------------------------------------------------------------------------
                                MAIN
  ---------------------------------------------------------------------------*/
int main()
{
   xmlctx     *ctx;
   oratext    *encoding, *doc;
   void       *saxcbctx;
   const xmlsaxcb *saxcb;
   uword       ecode;
   ub4         flags;

   encoding = doc = (oratext *) 0;
   saxcbctx = (void *) 0;
   saxcb = (const xmlsaxcb *) 0;
   flags = XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE;
   doc = (oratext *)DOCUMENT;

   /* initialize LPX context */
   if (!(ctx = xmlinit(&ecode, encoding,
                       (void (*)(void *, const oratext *, uword)) 0,
                       (void *) 0, saxcb, saxcbctx, (const xmlmemcb *) 0,
                       (void *) 0, (const oratext *) 0)))
   {
      printf("Failed to initialize XML parser, error %u\n", (unsigned) ecode);
      return -1;
   }

   /* parse the document */
```

```
   printf("\nParsing '%s' ...\n", doc);
   ecode = xmlparse(ctx, doc, encoding, flags);

   if (ecode)
     printf("Parse failed, code %u\n", (unsigned) ecode);
   else
     printf("Parse succeeded.\n");

   /* print results */

   printf("Printing results ...\n");
   DOMNSprint(ctx);

   /* terminate */

   (void) xmlterm(ctx);

   return (ecode ? -1 : 0);
}

/*-------------------------------------------------------------------------
                            DOMNSprint
  -------------------------------------------------------------------------*/
static void DOMNSprint(xmlctx *ctx)
{
    xmlnode *root;

    root = getDocumentElement(ctx);
    printf("\nThe elements are:\n");
    printElements(ctx, root);
}

/*-------------------------------------------------------------------------
                            printElements
  -------------------------------------------------------------------------*/
static void printElements(xmlctx *ctx, xmlnode *n)
{
    xmlnodes *nodes;
    uword     i;
    size_t    nn;

    const oratext   *qname;
    const oratext   *namespace;
    const oratext   *local;
```

```
    const oratext   *prefix;

    if (n == (xmlnode*)NULL)
       return;

    if (nodes = getChildNodes(n))
    {
       for (nn = numChildNodes(nodes), i = 0; i < nn; i++)
       {
          /* get node qualified name, local name, namespace, and prefix */

          qname = namespace = local = prefix = (oratext*)" ";

          if (getNodeQualifiedName(n) != (oratext*)NULL)
             qname = getNodeQualifiedName(n);
            if (getNodePrefix(n) != (oratext*)NULL)
             prefix = getNodePrefix(n);

          if (getNodeLocal(n) != (oratext *)NULL)
             local = getNodeLocal(n);

          if (getNodeNamespace(n) != (oratext*)NULL)
             namespace = getNodeNamespace(n);

          printf("  ELEMENT Qualified Name: %s\n", qname);
          printf("  ELEMENT Prefix Name   : %s\n", prefix);
          printf("  ELEMENT Local Name    : %s\n", local);
          printf("  ELEMENT Namespace     : %s\n", namespace);

          printAttrs(ctx, n);
          printElements(ctx, (xmlnode *) getChildNode(nodes, i));
       }
    }
}

/*---------------------------------------------------------------------
                            printAttrs
  ---------------------------------------------------------------------*/
static void printAttrs(xmlctx *ctx, xmlnode *n)
{
    xmlnodes  *attrs;
    xmlnode   *a;
    uword      i;
    size_t     na;
```

```
    const oratext   *value;
    const oratext   *qname;
    const oratext   *namespace;
    const oratext   *local;
    const oratext   *prefix;

    if (attrs = getAttributes(n))
    {
       printf("\n    ATTRIBUTES: \n");
       for (na = numAttributes(attrs), i = 0; i < na; i++)
       {
          /* get attr qualified name, local name, namespace, and prefix */

          a = getAttributeIndex(attrs, i);

          qname = namespace = local = prefix = value = (oratext*)" ";

          if (getAttrQualifiedName(a) != (oratext*)NULL)
             qname = getAttrQualifiedName(a);
          if (getAttrNamespace(a) != (oratext*)NULL)
             namespace = getAttrNamespace(a);
          if (getAttrLocal(a) != (oratext*)NULL)
             local = getAttrLocal(a);
          if (getAttrPrefix(a) != (oratext*)NULL)
             prefix = getAttrPrefix(a);
          if (getAttrValue(a) != (oratext*)NULL)
             value = getAttrValue(a);

          printf("      %s = %s\n", qname, value);
          printf("      Namespace : %s\n", namespace);
          printf("      Local Name: %s\n", local);
          printf("      Prefix    : %s\n\n", prefix);
       }
    }
    printf("\n");
}
```

XML Parser for C Example 11: C — DOMNamespace.std

`DOMNamespace.std` shows the expected output from `DOMNamespace.c`.

```
Parsing 'NSExample.xml' ...
Parse succeeded.
```

```
Printing results ...

The elements are:
  ELEMENT Qualified Name: doc
  ELEMENT Prefix Name   :
  ELEMENT Local Name    : doc
  ELEMENT Namespace     : http://www.w3c.org

    ATTRIBUTES:
      nsprefix:a1 = v1
      Namespace : http://www.oracle.com
      Local Name: a1
      Prefix    : nsprefix

      xmlns = http://www.w3c.org
      Namespace :
      Local Name: xmlns
      Prefix    :

      xmlns:nsprefix = http://www.oracle.com
      Namespace :
      Local Name: nsprefix
      Prefix    : xmlns


  ELEMENT Qualified Name: child
  ELEMENT Prefix Name   :
  ELEMENT Local Name    : child
  ELEMENT Namespace     : http://www.w3c.org
```

XML Parser for C Example 12: C — SAXNamespace.c

This example contains the C source code for the `SAXNamespace.c`.

```
/* Copyright (c) Oracle Corporation 1999. All Rights Reserved. */

/**
 ** This file demonstrates a simple use of the Namespace extensions to
 ** the SAX APIs.
 **/

#include <stdio.h>

#ifndef ORATYPES
```

```
# include <oratypes.h>
#endif

#ifndef ORAXML_ORACLE
# include <oraxml.h>
#endif

#define DOCUMENT         "NSExample.xml"

/*---------------------------------------------------------------------
                          FUNCTION PROTOTYPES
  ---------------------------------------------------------------------*/

static int sax_startdocument(void *ctx);
static int sax_enddocument(void *ctx);
static int sax_endelement(void *ctx, const oratext *name);
static int sax_nsstartelement(void *ctx, const oratext *qname,
                              const oratext *local,
                              const oratext *namespace,
                              const struct xmlnodes *attrs);

/* SAX callback structure */

xmlsaxcb sax_callback = {
    sax_startdocument,
    sax_enddocument,
    0,
    sax_endelement,
    0,
    0,
    0,
    0,
    0,
    sax_nsstartelement,
    0, 0, 0, 0, 0, 0, 0, 0
};

/* SAX callback context */

typedef struct {
    xmlctx  *ctx;
    uword    depth;
} cbctx;
```

```
/*-------------------------------------------------------------------------
                                MAIN
  -------------------------------------------------------------------------*/

int main()
{
   xmlctx     *ctx;
   uword       i;
   oratext    *doc, *encoding;
   xmlsaxcb   *saxcb;
   cbctx       saxctx;
   void       *saxcbctx;
   ub4         flags;
   uword       ecode;

   doc = encoding = (oratext *)0;
   flags = XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE;

   doc = (oratext *)DOCUMENT;

   /* set up SAX callbacks */

   saxcb = &sax_callback;
   saxcbctx = (void *) &saxctx;

   /* initialize LPX context */
   if (!(ctx = xmlinit(&ecode, encoding,
                       (void (*)(void *, const oratext *, uword)) 0,
                       (void *) 0, saxcb, saxcbctx, (const xmlmemcb *) 0,
                       (void *) 0, (const oratext *) 0)))
   {
      printf("Failed to initialize XML parser, error %u\n", (unsigned) ecode);
      return -1;
   }

   /* parse the document */

   printf("\nParsing '%s' ...\n", doc);
   ecode = xmlparse(ctx, doc, encoding, flags);

   if (ecode)
     printf("\nParse failed, code %u\n", (unsigned) ecode);
   else
     printf("\nParse succeeded.\n");
```

```
   /* terminate */

   (void) xmlterm(ctx);

   return (ecode ? -1 : 0);
}


/*---------------------------------------------------------------------
                            SAX Interface
  ---------------------------------------------------------------------*/

static int sax_startdocument(void *ctx)
{
    printf("\nStartDocument\n\n");
    return 0;
}


static int sax_enddocument(void *ctx)
{
    printf("\nEndDocument\n");
    return 0;
}


static int sax_endelement(void *ctx, const oratext *name)
{
    printf("\nELEMENT Name  : %s\n", name);
    return 0;
}


static int sax_nsstartelement(void *ctx, const oratext *qname,
                              const oratext *local,
                              const oratext *namespace,
                              const struct xmlnodes *attrs)
{
    cbctx   *saxctx = (cbctx *) ctx;
    xmlnode *attr;
    size_t   i;

    const oratext *aqname;
    const oratext *aprefix;
    const oratext *alocal;
```

```
const oratext *anamespace;
const oratext *avalue;

/*
 * Use the functions getXXXQualifiedName(), getXXXLocalName(), and
 * getXXXNamespace() to get Namespace information.
 */

if (qname == (oratext*)NULL)
   qname = (oratext*)" ";
if (local == (oratext*)NULL)
   local = (oratext*)" ";
if (namespace == (oratext*)NULL)
   namespace = (oratext*)" ";

printf("ELEMENT Qualified Name: %s\n", qname);
printf("ELEMENT Local Name    : %s\n", local);
printf("ELEMENT Namespace     : %s\n", namespace);

if (attrs)
{
   for (i = 0; i < numAttributes(attrs); i++)
   {
      attr = getAttributeIndex(attrs, i);

      aqname = aprefix = alocal = anamespace = avalue = (oratext*)" ";

      if (getAttrQualifiedName(attr) != (oratext*)NULL)
         aqname = getAttrQualifiedName(attr);

      if (getAttrPrefix(attr) != (oratext*)NULL)
         aprefix = getAttrPrefix(attr);

      if (getAttrLocal(attr) != (oratext*)NULL)
         alocal = getAttrLocal(attr);

      if (getAttrNamespace(attr) != (oratext*)NULL)
         anamespace = getAttrNamespace(attr);

      if (getAttrValue(attr) != (oratext*)NULL)
         avalue = getAttrValue(attr);

      printf(" ATTRIBUTE Qualified Name   : %s\n", aqname);
      printf(" ATTRIBUTE Prefix           : %s\n", aprefix);
      printf(" ATTRIBUTE Local Name       : %s\n", alocal);
```

```
            printf(" ATTRIBUTE Namespace        : %s\n", anamespace);
            printf(" ATTRIBUTE Value            : %s\n", avalue);
            printf("\n");
          }
      }
      return 0;
}
```

XML Parser for C Example 13: C — SAXNamespace.std

`SAXNamespace.std` shows the expected output from `SAXNamespace.c`

```
Parsing 'NSExample.xml' ...

StartDocument

ELEMENT Qualified Name: doc
ELEMENT Local Name    : doc
ELEMENT Namespace     : http://www.w3c.org
 ATTRIBUTE Qualified Name   : nsprefix:a1
 ATTRIBUTE Prefix           : nsprefix
 ATTRIBUTE Local Name       : a1
 ATTRIBUTE Namespace        : http://www.oracle.com
 ATTRIBUTE Value            : v1

 ATTRIBUTE Qualified Name   : xmlns
 ATTRIBUTE Prefix           :
 ATTRIBUTE Local Name       : xmlns
 ATTRIBUTE Namespace        :
 ATTRIBUTE Value            : http://www.w3c.org

 ATTRIBUTE Qualified Name   : xmlns:nsprefix
 ATTRIBUTE Prefix           : xmlns
 ATTRIBUTE Local Name       : nsprefix
 ATTRIBUTE Namespace        :
 ATTRIBUTE Value            : http://www.oracle.com

ELEMENT Qualified Name: child
ELEMENT Local Name    : child
ELEMENT Namespace     : http://www.w3c.org

ELEMENT Name  : child
```

```
ELEMENT Name  : doc

EndDocument

Parse succeeded.
```

XML Parser for C Example 14: C — FullDOM.c

This example contains the C source code for `FullDOM.c`

```
/* Copyright (c) Oracle Corporation 1999, 2000. All Rights Reserved. */

/*
   NAME
     FullDOM.c

   DESCRIPTION
     Sample code to test full DOM interface
*/

#include <stdio.h>

#ifndef ORATYPES
# include <oratypes.h>
#endif

#ifndef ORAXML_ORACLE
# include <oraxml.h>
#endif

#define TEST_DOCUMENT(oratext *) "FullDOM.xml"

void dump(xmlnode *node, uword level);
void dumpnode(xmlnode *node, uword level);

static char *ntypename[] = {
    "0",
    "ELEMENT",
    "ATTRIBUTE",
    "TEXT",
    "CDATA",
    "ENTREF",
    "ENTITY",
    "PI",
```

```
    "COMMENT",
    "DOCUMENT",
    "DTD",
    "DOCFRAG",
    "NOTATION"
};

#define FAIL { puts("Failed!"); exit(1); }

int main()
{
    xmlctx     *ctx;
    xmldtd     *dtd;
    xmlnode    *doc, *elem, *node, *text, *pi, *comment, *entref,
*subelem, *subtext, *cdata, *attr1, *attr2, *clone,
*deep_clone, *frag, *fragelem, *fragtext, *sub2,
*fish, *food, *gleep1, *gleep2, *repl;
    xmlnodes   *subs, *nodes, *attrs, *notes, *entities;
    uword       i, ecode, level;

    puts("XML C Full DOM test");

    puts("Initializing XML parser...");

    if (!(ctx = xmlinit(&ecode, (const oratext *) 0,
                        (void (*)(void *, const oratext *, uword)) 0,
                        (void *) 0, (const xmlsaxcb *) 0, (void *) 0,
(const xmlmemcb *) 0, (void *) 0,
                        (const oratext *) 0)))
    {
printf("Failed to initialze XML parser, error %u\n", (unsigned) ecode);
return 1;
    }

    puts("\nCreating new document...");
    if (!(doc = createDocument(ctx)))
FAIL

    puts("Document from root node:");
    dump(getDocument(ctx), 0);

    puts("\nCreating 'ROOT' element...");
    if (!(elem = createElement(ctx, (oratext *) "ROOT")))
FAIL
```

```
    puts("Setting as 'ROOT' element...");
    if (!appendChild(ctx, doc, elem))
FAIL

    puts("Document from 'ROOT' element:");
    dump(getDocumentElement(ctx), 0);

    puts("Adding 7 children to 'ROOT' element...");
    if (!(text = createTextNode(ctx, (oratext *) "Gibberish")) ||
        !appendChild(ctx, elem, text))
FAIL

    if (!(comment = createComment(ctx, (oratext*) "Bit warm today, innit?")) ||
        !appendChild(ctx, elem, comment))
FAIL

    if (!(pi = createProcessingInstruction(ctx, (oratext *) "target",
(oratext *) "PI-contents")) ||
        !appendChild(ctx, elem, pi))
FAIL

    if (!(cdata = createCDATASection(ctx, (oratext *) "See DATA")) ||
        !appendChild(ctx, elem, cdata))
FAIL

    if (!(entref = createEntityReference(ctx, (oratext *) "EntRef")) ||
        !appendChild(ctx, elem, entref))
FAIL

    if (!(fish = createElement(ctx, (oratext *) "FISH")) ||
!appendChild(ctx, elem, fish))
FAIL

    if (!(food = createElement(ctx, (oratext *) "FOOD")) ||
!appendChild(ctx, elem, food))
FAIL

    puts("Document from 'ROOT' element with its 7 children:");
    dump(getDocumentElement(ctx), 0);

    puts("\nTesting node insertion...");
    puts("Adding 'Pre-Gibberish' text node and 'Ask about the weather' comment
node ...");
    if (!(node = createTextNode(ctx, (oratext *) "Pre-Gibberish")) ||
        !insertBefore(ctx, elem, node, text))
```

```
FAIL

    if (!(node = createComment(ctx, (oratext *) "Ask about the weather:")) ||
        !insertBefore(ctx, elem, node, comment))
FAIL

    puts("Document from 'ROOT' element:");
    dump(getDocumentElement(ctx), 0);

    puts("\nTesting node removal by name ...");
    puts("Removing 'FISH' element");
    if (!(nodes = getChildNodes(elem)) ||
!removeNamedItem(nodes, (oratext *) "FISH"))
FAIL

    puts("Document from 'ROOT' element:");
    dump(getDocumentElement(ctx), 0);

    puts("\nTesting nextSibling links starting at first child...");
    for (node = getFirstChild(elem); node; node = getNextSibling(node))
dump(node, 1);

    puts("\nTesting previousSibling links starting at last child...");
    for (node = getLastChild(elem); node; node = getPreviousSibling(node))
dump(node, 1);

    puts("\nTesting setting node value...");
    puts("Original node:");
    dump(pi, 1);
    setNodeValue(pi, (oratext *) "New PI contents");
    puts("Node after new value:");
    dump(pi, 1);

    puts("\nAdding another element level, i.e., 'SUB' ...");
    if (!(subelem = createElement(ctx, (oratext *) "SUB")) ||
!insertBefore(ctx, elem, subelem, cdata) ||
!(subtext = createTextNode(ctx, (oratext *) "Lengthy SubText")) ||
        !appendChild(ctx, subelem, subtext))
FAIL

    puts("Document from 'ROOT' element:");
    dump(getDocumentElement(ctx), 0);

    puts("\nAdding a second 'SUB' element...");
    if (!(sub2 = createElement(ctx, (oratext *) "SUB")) ||
```

```
!insertBefore(ctx, elem, sub2, cdata))
FAIL

    puts("Document from 'ROOT' element:");
    dump(getDocumentElement(ctx), 0);

    puts("\nGetting all SUB nodes - note the distinct hex addresses ...");
    if (!(subs = getElementsByTagName(ctx, (xmlnode *) 0, (oratext *) "SUB")))
FAIL
    for (i = 0; i < getNodeMapLength(subs); i++)
dumpnode(getChildNode(subs, i), 1);

    puts("\nTesting parent links...");
    for (level = 1, node = subtext; node; node = getParentNode(node), level++)
dumpnode(node, level);

    puts("\nTesting owner document of node...");
    dumpnode(subtext, 1);
    dumpnode(getOwnerDocument(subtext), 1);

    puts("\nTesting node replacement...");
    if (!(node = createTextNode(ctx, (oratext *) "REPLACEMENT, 1/2 PRICE")) ||
        !replaceChild(ctx, pi, node))
FAIL

    puts("Document from 'ROOT' element:");
    dump(getDocumentElement(ctx), 0);

    puts("\nTesting node removal...");
    if (!removeChild(entref))
FAIL

    puts("Document from 'ROOT' element:");
    dump(getDocumentElement(ctx), 0);

    puts("\nNormalizing...");
    normalize(ctx, elem);

    puts("Document from 'ROOT' element:");
    dump(getDocumentElement(ctx), 0);

    puts("\nCreating and populating document fragment...");
    if (!(frag = createDocumentFragment(ctx)) ||
!(fragelem = createElement(ctx, (oratext *) "FragElem")) ||
!(fragtext = createTextNode(ctx, (oratext *) "FragText")) ||
```

```
!appendChild(ctx, frag, fragelem) ||
!appendChild(ctx, frag, fragtext))
FAIL
    dump(frag, 1);

    puts("Insert document fragment...");
    if (!insertBefore(ctx, elem, frag, comment))
FAIL
    dump(elem, 1);

    puts("\nCreate two attributes...");
    if (!(attr1 = createAttribute(ctx,(oratext*)"Attr1",(oratext*)"Value1")) ||
!(attr2 = createAttribute(ctx,(oratext*)"Attr2",(oratext*)"Value2")))
FAIL
    puts("Setting attributes...");
    if (!setAttributeNode(ctx, subelem, attr1, NULL) ||
!setAttributeNode(ctx, subelem, attr2, NULL))
FAIL
    dump(subelem, 1);

    puts("\nAltering attribute1 value...");
    setAttrValue(attr1, (oratext *) "New1");
    dump(subelem, 1);

    puts("\nFetching attribute by name (Attr2)...");
    if (!(node = getAttributeNode(subelem, (oratext *) "Attr2")))
FAIL
    dump(node, 1);

    puts("\nRemoving attribute by name (Attr1)...");
    removeAttribute(subelem, (oratext *) "Attr1");
    dump(subelem, 1);

    puts("\nAdding new attribute...");
    if (!setAttribute(ctx, subelem, (oratext *) "Attr3", (oratext *) "Value3"))
FAIL
    dump(subelem, 1);

    puts("\nRemoving attribute by pointer (Attr2)...");
    if (!removeAttributeNode(subelem, attr2))
FAIL
    dump(subelem, 1);

    puts("\nAdding new attribute w/same name (test replacement)...");
    dump(subelem, 1);
```

```
    if (!(node = createAttribute(ctx, (oratext*)"Attr3", (oratext*)"Zoo3")))
FAIL
    if (!setAttributeNode(ctx, subelem, node, NULL))
FAIL
    dump(subelem, 1);

    puts("\nTesting node (attr) set by name ...");
    puts("Adding 'GLEEP' attribute and printing out hex addresses of node set");
    attrs = getAttributes(subelem);
    if (!(gleep1=createAttribute(ctx,(oratext*)"GLEEP",(oratext*)"gleep1")) ||
!setNamedItem(ctx, attrs, gleep1, NULL))
FAIL
    dump(subelem, 1);

    puts("\nTesting node set by name ...");
    puts("Replacing 'GLEEP' attribute - note the changed hex address");
    if (!(gleep2=createAttribute(ctx,(oratext*)"GLEEP",(oratext*)"gleep2")) ||
!setNamedItem(ctx, attrs, gleep2, &repl))
FAIL
    dump(subelem, 1);
    puts("Replaced node was:");
    dump(repl, 1);

    puts("\nOriginal SubROOT...");
    dump(subelem, 1);
    puts("Cloned SubROOT (not deep)...");
    clone = cloneNode(ctx, subelem, FALSE);
    dump(clone, 1);
    puts("Cloned SubROOT (deep)...");
    deep_clone = cloneNode(ctx, subelem, TRUE);
    dump(deep_clone, 1);

    puts("\nSplitting text...");
    dump(subelem, 1);
    splitText(ctx, subtext, 3);
    dump(subelem, 1);

    puts("\nTesting string operations...");
    printf("    CharData = \"%s\"\n", getCharData(subtext));
    puts("Setting new data...");
    setCharData(subtext, (oratext *) "0123456789");
    printf("    CharData = \"%s\"\n", getCharData(subtext));
    printf("    CharLength = %d\n", getCharLength(subtext));
    printf("    Substring(0,5) = \"%s\"\n",
substringData(ctx, subtext, 0, 5));
```

```
    printf("    Substring(8,2) = \"%s\"\n",
substringData(ctx, subtext, 8, 2));
    puts("Appending data...");
    appendData(ctx, subtext, (oratext *) "ABCDEF");
    printf("    CharData = \"%s\"\n", getCharData(subtext));
    puts("Inserting data...");
    insertData(ctx, subtext, 10, (oratext *) "*foo*");
    printf("    CharData = \"%s\"\n", getCharData(subtext));
    puts("Deleting data...");
    deleteData(ctx, subtext, 0, 10);
    printf("    CharData = \"%s\"\n", getCharData(subtext));
    puts("Replacing data...");
    replaceData(ctx, subtext, 1, 3, (oratext *) "bamboozle");
    printf("    CharData = \"%s\"\n", getCharData(subtext));

    puts("Cleaning up...");
    xmlclean(ctx);

    if (getDocument(ctx))
    {
puts("Problem, document is not gone!!");
return 1;
    }

    puts("Parsing test document...");
    if (ecode = xmlparse(ctx, TEST_DOCUMENT, (oratext *) 0, 0))
    {
printf("Parse failed, code %d\n", (int) ecode);
return 1;
    }

    puts("Document from root node:");
    dump(getDocument(ctx), 0);

    dtd = getDocType(ctx);

    puts("Testing getDocTypeNotations...");
    if (notes = getDocTypeNotations(dtd))
    {
size_t n_notes = numChildNodes(notes);

printf("# of notations = %d\n", (int) n_notes);
for (i = 0; i < n_notes; i++)
    dump(getChildNode(notes, i), 1);
    }
```

```
    else
puts("No defined notations\n");

    puts("Testing getDocTypeEntities...");
    if (entities = getDocTypeEntities(dtd))
    {
size_t n_entities = numChildNodes(entities);

printf("# of entities = %d\n", (int) n_entities);
for (i = 0; i < n_entities; i++)
    dump(getChildNode(entities, i), 1);
    }
    else
puts("No defined entities\n");

    puts("Cleaning up...");
    xmlclean(ctx);

    if (getDocument(ctx))
    {
puts("Problem, document is not gone!!\n");
return 1;
    }

    puts("\nTerminating parser...");
    xmlterm(ctx);

    puts("Success.");
    return 0;
}

void dump(xmlnode *node, uword level)
{
    xmlnodes *nodes;
    uword     i, n_nodes;

    if (node)
    {
dumpnode(node, level);
if (hasChildNodes(node))
{
    nodes = getChildNodes(node);
    n_nodes = numChildNodes(nodes);
    for (i = 0; i < n_nodes; i++)
dump(getChildNode(nodes, i), level + 1);
```

```
}
   }


void dumpnode(xmlnode *node, uword level)
{
    const oratext  *name, *value;
    xmlntype  type;
    xmlnodes *attrs;
    xmlnode  *attr;
    uword     i, n_attrs;

    if (node)
    {
for (i = 0; i <= level; i++)
    fputs("    ", stdout);
type = getNodeType(node);
fputs(ntypename[type], stdout);
if ((name = getNodeName(node)) && (*name != '#'))
    printf(" \"%s\"", (char *) name);
if (value = getNodeValue(node))
    printf(" = \"%s\"", (char *) value);
if ((type == ELEMENT_NODE) && (attrs = getAttributes(node)))
{
    fputs(" [", stdout);
    n_attrs = numAttributes(attrs);
    for (i = 0; i < n_attrs; i++)
    {
if (i) fputs(", ", stdout);
attr = getAttributeIndex(attrs, i);
fputs((char *) getAttrName(attr), stdout);
if (getAttrSpecified(attr))
    putchar('*');
printf("=\"%s\"", (char *) getAttrValue(attr));
    }
    putchar(']');
}
putchar('\n');
    }
}

/* end of FullDOM.c */
```

XML Parser for C Example 15: C — FullDOM.std

`FullDOM.std` shows the expected output from `FullDOM.c`.

```
XML C Full DOM test
Initializing XML parser...

Creating new document...
Document from root node:
    DOCUMENT

Creating 'ROOT' element...
Setting as 'ROOT' element...
Document from 'ROOT' element:
    ELEMENT "ROOT"
Adding 7 children to 'ROOT' element...
Document from 'ROOT' element with its 7 children:
    ELEMENT "ROOT"
        TEXT = "Gibberish"
        COMMENT = "Bit warm today, innit?"
        PI "target" = "PI-contents"
        CDATA = "See DATA"
        ENTREF "EntRef"
        ELEMENT "FISH"
        ELEMENT "FOOD"

Testing node insertion...
Adding 'Pre-Gibberish' text node and 'Ask about the weather' comment node ...
Document from 'ROOT' element:
    ELEMENT "ROOT"
        TEXT = "Pre-Gibberish"
        TEXT = "Gibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        PI "target" = "PI-contents"
        CDATA = "See DATA"
        ENTREF "EntRef"
        ELEMENT "FISH"
        ELEMENT "FOOD"

Testing node removal by name ...
Removing 'FISH' element
Document from 'ROOT' element:
    ELEMENT "ROOT"
        TEXT = "Pre-Gibberish"
```

```
        TEXT = "Gibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        PI "target" = "PI-contents"
        CDATA = "See DATA"
        ENTREF "EntRef"
        ELEMENT "FOOD"

Testing nextSibling links starting at first child...
        TEXT = "Pre-Gibberish"
        TEXT = "Gibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        PI "target" = "PI-contents"
        CDATA = "See DATA"
        ENTREF "EntRef"
        ELEMENT "FOOD"

Testing previousSibling links starting at last child...
        ELEMENT "FOOD"
        ENTREF "EntRef"
        CDATA = "See DATA"
        PI "target" = "PI-contents"
        COMMENT = "Bit warm today, innit?"
        COMMENT = "Ask about the weather:"
        TEXT = "Gibberish"
        TEXT = "Pre-Gibberish"

Testing setting node value...
Original node:
        PI "target" = "PI-contents"
Node after new value:
        PI "target" = "New PI contents"

Adding another element level, i.e., 'SUB' ...
Document from 'ROOT' element:
    ELEMENT "ROOT"
        TEXT = "Pre-Gibberish"
        TEXT = "Gibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        PI "target" = "New PI contents"
        ELEMENT "SUB"
            TEXT = "Lengthy SubText"
        CDATA = "See DATA"
```

```
        ENTREF "EntRef"
        ELEMENT "FOOD"

Adding a second 'SUB' element...
Document from 'ROOT' element:
    ELEMENT "ROOT"
        TEXT = "Pre-Gibberish"
        TEXT = "Gibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        PI "target" = "New PI contents"
        ELEMENT "SUB"
            TEXT = "Lengthy SubText"
        ELEMENT "SUB"
        CDATA = "See DATA"
        ENTREF "EntRef"
        ELEMENT "FOOD"

Getting all SUB nodes - note the distinct hex addresses ...
        ELEMENT "SUB"
        ELEMENT "SUB"

Testing parent links...
        TEXT = "Lengthy SubText"
            ELEMENT "SUB"
                ELEMENT "ROOT"
                    DOCUMENT

Testing owner document of node...
        TEXT = "Lengthy SubText"
        DOCUMENT

Testing node replacement...
Document from 'ROOT' element:
    ELEMENT "ROOT"
        TEXT = "Pre-Gibberish"
        TEXT = "Gibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        TEXT = "REPLACEMENT, 1/2 PRICE"
        ELEMENT "SUB"
            TEXT = "Lengthy SubText"
        ELEMENT "SUB"
        CDATA = "See DATA"
        ENTREF "EntRef"
```

```
        ELEMENT "FOOD"

Testing node removal...
Document from 'ROOT' element:
    ELEMENT "ROOT"
        TEXT = "Pre-Gibberish"
        TEXT = "Gibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        TEXT = "REPLACEMENT, 1/2 PRICE"
        ELEMENT "SUB"
            TEXT = "Lengthy SubText"
        ELEMENT "SUB"
        CDATA = "See DATA"
        ELEMENT "FOOD"

Normalizing...
Document from 'ROOT' element:
    ELEMENT "ROOT"
        TEXT = "Pre-GibberishGibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        TEXT = "REPLACEMENT, 1/2 PRICE"
        ELEMENT "SUB"
            TEXT = "Lengthy SubText"
        ELEMENT "SUB"
        CDATA = "See DATA"
        ELEMENT "FOOD"

Creating and populating document fragment...
        DOCFRAG
            ELEMENT "FragElem"
            TEXT = "FragText"
Insert document fragment...
        ELEMENT "ROOT"
            TEXT = "Pre-GibberishGibberish"
            COMMENT = "Ask about the weather:"
            ELEMENT "FragElem"
            TEXT = "FragText"
            COMMENT = "Bit warm today, innit?"
            TEXT = "REPLACEMENT, 1/2 PRICE"
            ELEMENT "SUB"
                TEXT = "Lengthy SubText"
            ELEMENT "SUB"
            CDATA = "See DATA"
```

```
            ELEMENT "FOOD"

Create two attributes...
Setting attributes...
        ELEMENT "SUB" [Attr1*="Value1", Attr2*="Value2"]
            TEXT = "Lengthy SubText"

Altering attribute1 value...
        ELEMENT "SUB" [Attr1*="New1", Attr2*="Value2"]
            TEXT = "Lengthy SubText"

Fetching attribute by name (Attr2)...
        ATTRIBUTE "Attr2" = "Value2"

Removing attribute by name (Attr1)...
        ELEMENT "SUB" [Attr2*="Value2"]
            TEXT = "Lengthy SubText"

Adding new attribute...
        ELEMENT "SUB" [Attr2*="Value2", Attr3*="Value3"]
            TEXT = "Lengthy SubText"

Removing attribute by pointer (Attr2)...
        ELEMENT "SUB" [Attr3*="Value3"]
            TEXT = "Lengthy SubText"

Adding new attribute w/same name (test replacement)...
        ELEMENT "SUB" [Attr3*="Value3"]
            TEXT = "Lengthy SubText"
        ELEMENT "SUB" [Attr3*="Zoo3"]
            TEXT = "Lengthy SubText"

Testing node (attr) set by name ...
Adding 'GLEEP' attribute and printing out hex addresses of node set
        ELEMENT "SUB" [Attr3*="Zoo3", GLEEP*="gleep1"]
            TEXT = "Lengthy SubText"

Testing node set by name ...
Replacing 'GLEEP' attribute - note the changed hex address
        ELEMENT "SUB" [Attr3*="Zoo3", GLEEP*="gleep2"]
            TEXT = "Lengthy SubText"
Replaced node was:
        ATTRIBUTE "GLEEP" = "gleep1"

Original SubROOT...
```

```
        ELEMENT "SUB" [Attr3*="Zoo3", GLEEP*="gleep2"]
            TEXT = "Lengthy SubText"
Cloned SubROOT (not deep)...
        ELEMENT "SUB" [Attr3*="Zoo3", GLEEP*="gleep2"]
            TEXT = "Lengthy SubText"
Cloned SubROOT (deep)...
        ELEMENT "SUB" [Attr3*="Zoo3", GLEEP*="gleep2"]
            TEXT = "Lengthy SubText"

Splitting text...
        ELEMENT "SUB" [Attr3*="Zoo3", GLEEP*="gleep2"]
            TEXT = "Lengthy SubText"
        ELEMENT "SUB" [Attr3*="Zoo3", GLEEP*="gleep2"]
            TEXT = "Leng"
            TEXT = "thy SubText"

Testing string operations...
    CharData = "Leng"
Setting new data...
    CharData = "0123456789"
    CharLength = 10
    Substring(0,5) = "01234"
    Substring(8,2) = "89"
Appending data...
    CharData = "0123456789ABCDEF"
Inserting data...
    CharData = "0123456789*foo*ABCDEF"
Deleting data...
    CharData = "*foo*ABCDEF"
Replacing data...
    CharData = "*bamboozle*ABCDEF"
Cleaning up...
Parsing test document...
Document from root node:
    DOCUMENT
        DTD "doc"
        ELEMENT "doc" [xml:lang*="foo"]
            ELEMENT "p" [xml:space="preserve"]
                TEXT = "An ampersand (&) may be escaped
numerically (&#38;) or with a general entity
(&amp;)."
Testing getDocTypeNotations...
# of notations = 2
        NOTATION "notation1"
        NOTATION "notation2"
```

```
Testing getDocTypeEntities...
# of entities = 1
        ENTITY "example" = "<p>An ampersand (&#38;) may be escaped
numerically (&#38;#38;) or with a general entity
(&amp;amp;).</p>"
Cleaning up...

Terminating parser...
Success.
```

XML Parser for C Example 16: C — XSLSample.c

This example contains C source code for `XSLSample.c`.

```
/* Copyright (c) Oracle Corporation 1999. All Rights Reserved. */

/*
   NAME
     XSLSample.c - Sample function for XSL
   DESCRIPTION
     Sample usage of C XSL Processor
*/

#include <stdio.h>
#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;
}
```

XML Parser for C Example 17: C — XSLSample.std

`XSLSample.std` shows the expected output from `XSLSample.c`.

```
Parsing 'class.xml' ...
Parsing 'iden.xsl' ...
XSL Processing
<root>
   <course>
      <Name>Calculus</Name>
      <Dept>Math</Dept>
      <Instructor>
         <Name>Jim Green</Name>
      </Instructor>
      <Student>
         <Name>Jack</Name>
         <Name>Mary</Name>
         <Name>Paul</Name>
      </Student>
   </course>
</root>
```

PartVIII

XDK for C++

Part VIII describes how to access and use XML Development Kit (XDK) for C++.

It contains the following chapters:

22

Using XML Parser for C++

This chapter contains the following sections:

Accessing XML Parser for C++

The XML Parser for C++ is provided with Oracle8*i* and is also available for download from the OTN site: http://technet.oracle.com/tech/xml.

It is located at $ORACLE_HOME/xdk/cpp/parser.

XML Parser for C++ Features

`readme.html` in the root directory of the software archive contains release specific information including bug fixes and API additions.

XML Parser for C++ will check if an XML document is well-formed, and optionally validate it against a DTD. The parser will construct an object tree which can be accessed via a DOM interface or operate serially via a SAX interface.

You can post questions, comments, or bug reports to the XML Discussion Forum at http://technet.oracle.com.

Specifications

See Appendix E, "XDK for C++: Specifications and Cheat Sheet" for a list of XML Parser for C++ specifications and methods.

See Also:

- The doc directory in your install area
- *Oracle8i XML Reference*
- On OTN under http://technet.oracle.com/tech/xml/parser_cpp/index.htm

Memory Allocation

The memory callback functions `memcb` may be used if you wish to use your own memory allocation. If they are used, all of the functions should be specified.

The memory allocated for parameters passed to the SAX callbacks or for nodes and data stored with the DOM parse tree will not be freed until one of the following is done:

- `xmlparse()` or `xmlparsebuf()`is called to parse another file or buffer.
- `xmlclean()` is called.
- `xmlterm()` is called.

Thread Safety

If threads are forked off somewhere in the midst of the init-parse-term sequence of calls, you will get unpredictable behavior and results.

Data Types Index

Table 22–1 lists the datatypes used in XML Parser for C++.

Table 22–1 Datatypes Used in XML Parser for C++

DataType	Description
oratext	String pointer
xmlctx	Master XML context
xmlmemcb	Memory callback structure (optional)
xmlsaxcb	SAX callback structure (SAX only)
ub4	32-bit (or larger) unsigned integer
uword	Native unsigned integer

Error Message Files

Error message files are provided in the mesg/ subdirectory. The messages files also exist in the `$ORACLE_HOME/oracore/mesg` directory. You may set the environment variable `ORA_XML_MESG` to point to the absolute path of the `mesg/` subdirectory although this not required.

XML Parser for C++ Usage

Figure 22–1 illustrates the XML Parser for C++ functionality.

1. The parsing process begins with the `xmlinit()` method.
2. The XML input can be either an XML file or string buffer. This inputs the following methods:
   - `XMLParser.xmlparse()` if the input is an XML file
   - `XMLParser.xmlparseBuffer()` if the input is a string buffer
3. DOM or SAX API

   DOM: If you are using the DOM interface, include the following steps:

   - The `XMLParser.xmlparse()` or `.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 optionally first invoke `.xmlclean()` to clean up any data structure created during the parse process. You would then call `.xmlterm()`

   SAX: If you are using the SAX interface, include the following steps:

   - Process the results of the parser from `.xmlparse()` or `.xmlparseBuffer()` via callback methods.
   - Register the callback methods
4. Optionally, use `.xmlclean()` to clean up the memory and structures used during a parse, and go to Step 5. or return to Step 2.
5. Terminate the parsing process with `.xmlterm()`

Parser Calling Sequence

The sequence of calls to the parser can be any of the following:

- `XMLParser.xmlinit()` - `XMLParser.xmlparse()` or

  `XMLParser.xmlparsebuf()` - `XMLParser.xmlterm()`
- `XMLParser.xmlinit()` - `XMLParser.xmlparse()` or

```
XMLParser.xmlparsebuf() - XMLParser.xmlclean() -
XMLParser.xmlparse()
```
or

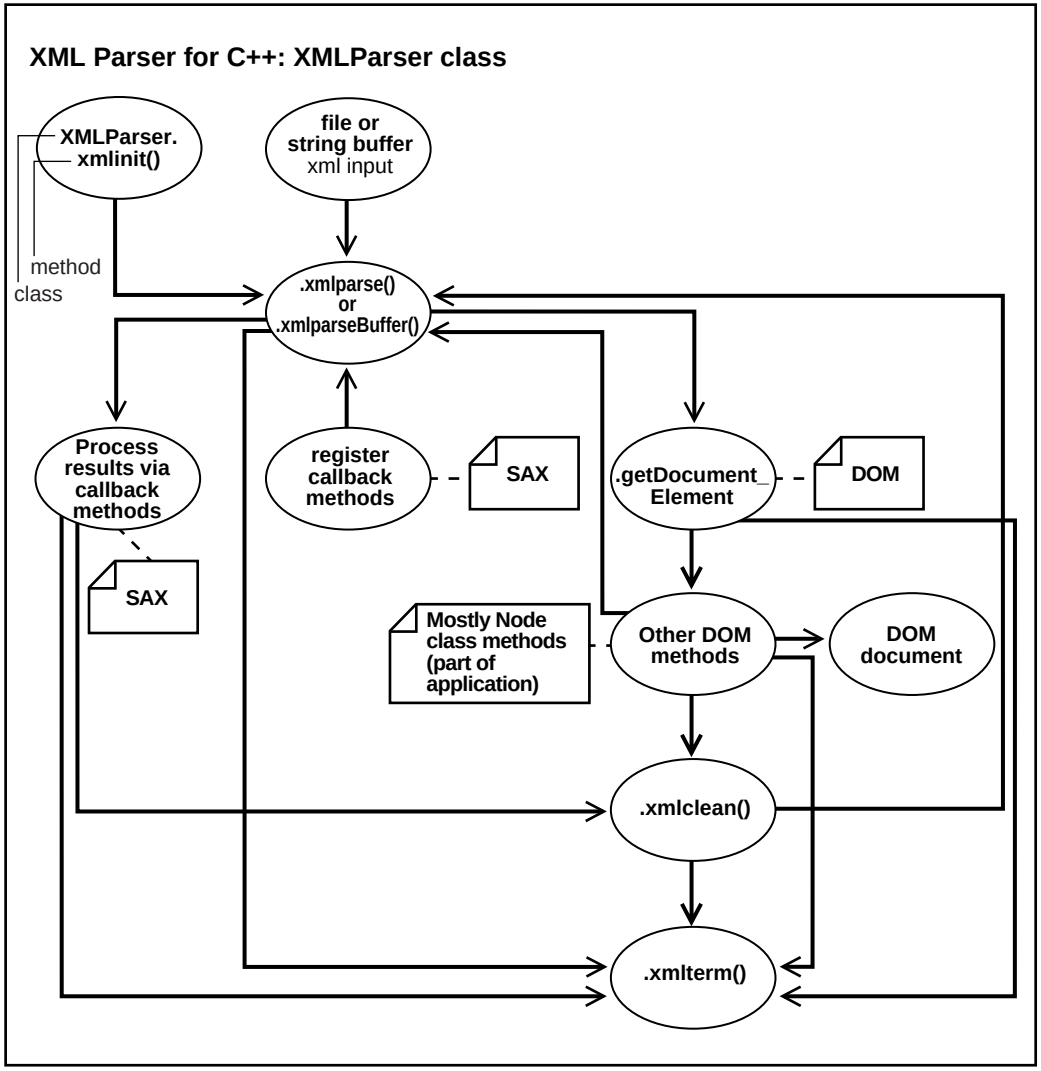
```
XMLParser.xmlparsebuf() - XMLParser.xmlclean() - ... -
XMLParser.xmlterm()
```

- `XMLParser.xmlinit() - XMLParser.xmlparse()` or

  `XMLParser.xmlparsebuf() - XMLParser.xmlparse()` or

  `XMLParser.xmlparsebuf() - ... - XMLParser.xmlterm()`

Figure 22–1 XML Parser for C++ (DOM and SAX Interfaces) Usage



XML Parser for C++ XSLT (DOM Interface) Usage

Figure 22–2 shows the XML Parser for C++ XSLT functionality for the DOM interface.

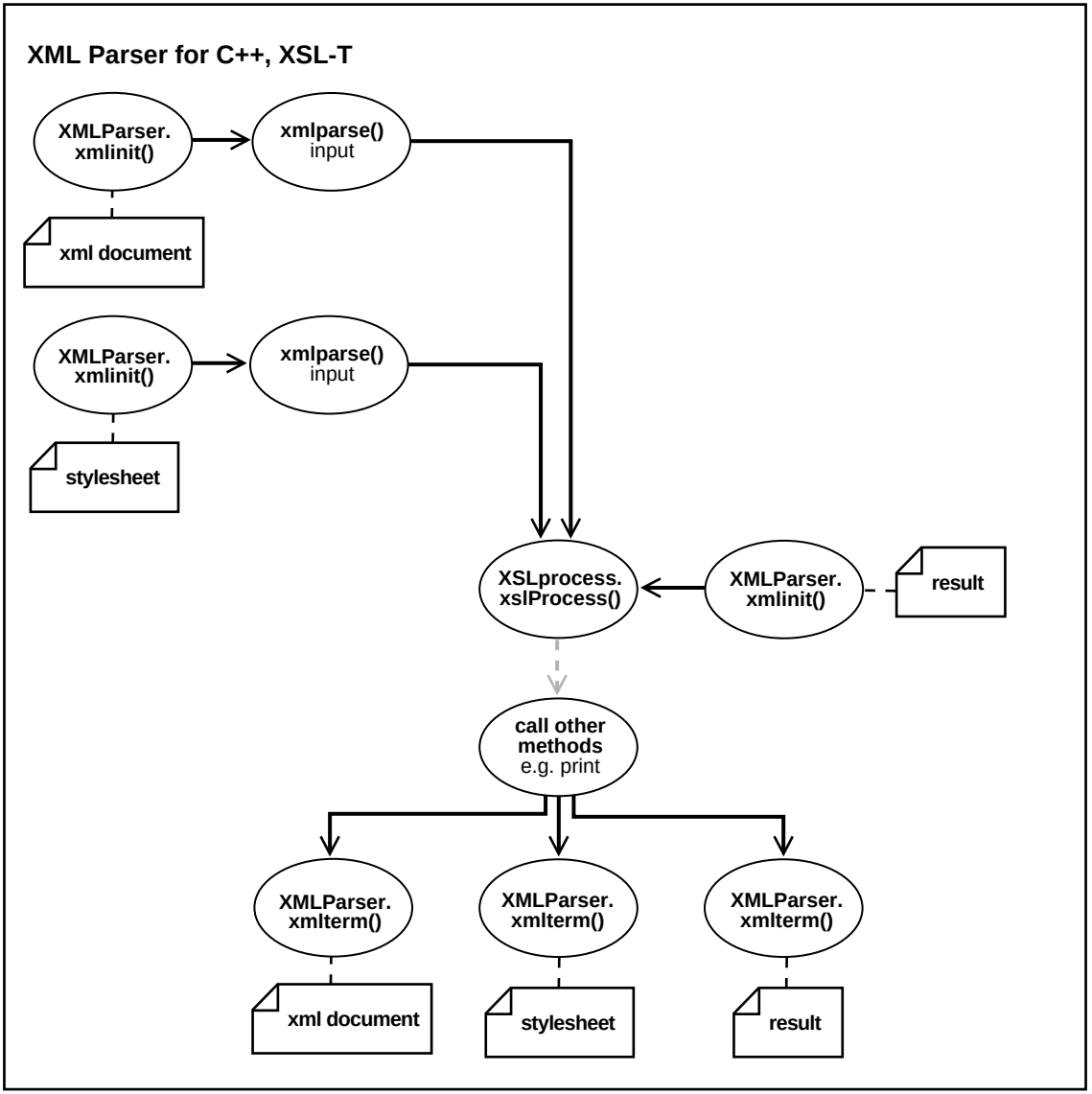
1. There are two inputs to XMLParser.`xmlparse()`:
   - The Stylesheet to be applied to the XML document
   - XML document

   The output of XMLParser.`xmlparse()`, the parsed stylesheet and parsed XML document, are sent to the XSLProcess.`xslprocess()` method for processing.
2. `XMLParser.xmlinit()` initializes the XSLT processing. XMLParser.`xmlinit() also` initializes the `xslprocess()` result
3. `XSLProcess.xslProcess()` optionally calls other methods, such as print methods. You can see the list of available methods either on OTN or in *Oracle8i XML Reference*.
4. The resultant document (XML, HTML, VML, and so on) is typically sent to an application for further processing.
5. The application terminates the XSLT process by declaring `XMLParser.xmlterm()`, for the XML document, stylesheet, and final result.

XML Parser for C XSLT functionality is illustrated with the following examples:

- XML Parser for C++ Example 16: C++ — XSLSample.cpp on page 22-54
- XML Parser for C++ Example 17: C++ — XSLSample.std on page 22-56

Figure 22–2 Parser for C++: XSL-T Functionality (DOM Interface) Usage



Default Behavior

The following is the XML Parser for C++ default behavior:

- Character set encoding is UTF-8. If all your documents are ASCII, you are encouraged to set the encoding to US-ASCII for better performance.
- Messages are printed to stderr unless msghdlr is given.
- A parse tree which can be accessed by DOM APIs is built unless saxcb is set to use the SAX callback APIs. Note that any of the SAX callback functions can be set to NULL if not needed.
- The default behavior for the parser is to check that the input is well-formed but not to check whether it is valid. The flag XML_FLAG_VALIDATE can be set to validate the input. The default behavior for whitespace processing is to be fully conformant to the XML 1.0 spec, that is, all whitespace is reported back to the application but it is indicated which whitespace is ignorable. However, some applications may prefer to set the XML_FLAG_DISCARD_WHITESPACE which will discard all whitespace between an end-element tag and the following start-element tag.

Note: It is recommended that you set the default encoding explicitly if using only single byte character sets (such as US-ASCII or any of the ISO-8859 character sets) for performance up to 25% faster than with multibyte character sets, such as UTF-8.

DOM and SAX APIs

Oracle XML parser for C++ checks if an XML document is well-formed, and optionally validates it against a DTD. The parser constructs an object tree which can be accessed via one of the following interfaces:

- DOM interface
- Serially via a SAX interface

These two XML APIs:

- DOM: Tree-based APIs. A tree-based API compiles an XML document into an internal tree structure, then allows an application to navigate that tree using the Document Object Model (DOM), a standard tree-based API for XML and HTML documents.
- SAX: Event-based APIs. An event-based API, on the other hand, reports parsing events (such as the start and end of elements) directly to the application through callbacks, and does not usually build an internal tree. The application implements handlers to deal with the different events, much like handling events in a graphical user interface.

Tree-based APIs are useful for a wide range of applications, but they often put a great strain on system resources, especially if the document is large (under very controlled circumstances, it is possible to construct the tree in a lazy fashion to avoid some of this problem). Furthermore, some applications need to build their own, different data trees, and it is very inefficient to build a tree of parse nodes, only to map it onto a new tree.

In both of these cases, an event-based API provides a simpler, lower-level access to an XML document: you can parse documents much larger than your available system memory, and you can construct your own data structures using your callback event handlers.

Using the SAX API

To use SAX, an `xmlsaxcb` structure is initialized with function pointers and passed to the `xmlinit()` call. A pointer to a user-defined context structure can also be included. That context pointer will be passed to each SAX function.

SAX Callback Structure

The SAX callback structure:

```
typedef struct
```

```
{
 sword (*startDocument)(void *ctx);
 sword (*endDocument)(void *ctx);
 sword (*startElement)(void *ctx, const oratext *name,
            const struct xmlarray *attrs);
 sword (*endElement)(void *ctx, const oratext *name);
 sword (*characters)(void *ctx, const oratext *ch, size_t len);
 sword (*ignorableWhitespace)(void *ctx, const oratext *ch, size_t len);
 sword (*processingInstruction)(void *ctx, const oratext *target,
            const oratext *data);
 sword (*notationDecl)(void *ctx, const oratext *name,
            const oratext *publicId, const oratext *systemId);
 sword (*unparsedEntityDecl)(void *ctx, const oratext *name,
            const oratext *publicId,
            const oratext *systemId, const oratext *notationName);
 sword (*nsStartElement)(void *ctx, const oratext *qname,
            const oratext *local, const oratext *nsp,
            const struct xmlnodes *attrs);
} xmlsaxcb;
```

Using the DOM API

See "XML Parser for C++ Example 6: C++ — DOMSample.cpp" on page 22-18.

Invoking XML Parser for C++

XML Parser for C++ can be invoked in two ways:

- By invoking the executable on the command line
- By writing C++ code and using the supplied APIs

Command Line Usage

The XML Parser for C++ can be called as an executable by invoking `bin/xml`

Table 22–2 lists the command line options.

Table 22–2 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/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 22–3 lists the sample files in sample/ directory.

Table 22–3 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
SAXSample.cpp	Source for SAXSample program
SAXSample.std	Expected output from SAXSample
XSLSample.cpp	Source for XSLSample program
XSLSample.std	Expected output from XSLSample
class.xml	XML file that may be used with XSLSample
iden.xsl	Stylesheet that may be used with XSLSample
cleo.xml	XML version of Shakespeare's play

Running the XML Parser for C++ Sample Programs

Building the Sample programs

Change directories to ..sample/ and read the README file. This will explain how to build the sample programs according to your platform.

Sample Programs

Table 22–4 lists the programs built by the sample files in sample/.

Table 22–4 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, i.e. all the unique SPEAKER elements within each SCENE element.
DOMSample [speaker]	A sample application using DOM APIs. Prints all speeches made by the given speaker. If no speaker is specified, "Soothsayer" is used. Note that major characters have uppercase names (e.g. "CLEOPATRA"), whereas minor characters have capitalized names (e.g. "Attendant"). See the output of SAXSample.
SAXNamespace	A sample application using Namespace extensions to SAX API; prints out all elements and attributes of NSExample.xml along with full namespace information.
DOMNamespace	Same as SAXNamespace except using DOM interface.
FullDOM	Sample usage of full DOM interface. Exercises all the calls, but does nothing too exciting.
XSLSample <xmlfile> <xsl ss>	Sample usage of XSL processor. It takes two filenames as input, the XML file and the XSL stylesheet. Note: If you redirect stdout of this program to a file, you may encounter some missing output, depending on your environment.

XML Parser for C++ Example 1: XML — class.xml

`class.xml` is an XML file that inputs XSLSample.cpp.

```
<?xml version = "1.0"?>
<!DOCTYPE course [
<!ELEMENT course (Name, Dept, Instructor, Student)>
```

```
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Dept (#PCDATA)>
<!ELEMENT Instructor (Name)>
<!ELEMENT Student (Name*)>
]>
<course>
<Name>Calculus</Name>
<Dept>Math</Dept>
<Instructor>
<Name>Jim Green</Name>
</Instructor>
<Student>
<Name>Jack</Name>
<Name>Mary</Name>
<Name>Paul</Name>
</Student>
</course>
```

XML Parser for C++ Example 2: XML — cleo.xml

This XML example inputs `DOMSample.cpp` and `SAXSample.cpp`.

```
<?xml version="1.0"?>
<!DOCTYPE PLAY [
    <!ELEMENT PLAY     (TITLE, PERSONAE, SCNDESCR, PLAYSUBT, INDUCT?,
 PROLOGUE?, ACT+, EPILOGUE?)>
    <!ELEMENT TITLE    (#PCDATA)>
    <!ELEMENT FM       (P+)>
    <!ELEMENT P        (#PCDATA)>
    <!ELEMENT PERSONAE (TITLE, (PERSONA | PGROUP)+)>
    <!ELEMENT PGROUP   (PERSONA+, GRPDESCR)>
    <!ELEMENT PERSONA  (#PCDATA)>
    <!ELEMENT GRPDESCR (#PCDATA)>
    <!ELEMENT SCNDESCR (#PCDATA)>
    <!ELEMENT PLAYSUBT (#PCDATA)>
    <!ELEMENT INDUCT   (TITLE, SUBTITLE*, (SCENE+|(SPEECH|STAGEDIR|SUBHEAD)+))>
    <!ELEMENT ACT      (TITLE, SUBTITLE*, PROLOGUE?, SCENE+, EPILOGUE?)>
    <!ELEMENT SCENE    (TITLE, SUBTITLE*, (SPEECH | STAGEDIR | SUBHEAD)+)>
    <!ELEMENT PROLOGUE (TITLE, SUBTITLE*, (STAGEDIR | SPEECH)+)>
    <!ELEMENT EPILOGUE (TITLE, SUBTITLE*, (STAGEDIR | SPEECH)+)>
    <!ELEMENT SPEECH   (SPEAKER+, (LINE | STAGEDIR | SUBHEAD)+)>
    <!ELEMENT SPEAKER  (#PCDATA)>
    <!ELEMENT LINE     (#PCDATA | STAGEDIR)*>
    <!ELEMENT STAGEDIR (#PCDATA)>
```

```
    <!ELEMENT SUBTITLE (#PCDATA)>
    <!ELEMENT SUBHEAD  (#PCDATA)>
]>

<PLAY>
<TITLE>The Tragedy of Antony and Cleopatra</TITLE>

<PERSONAE>
<TITLE>Dramatis Personae</TITLE>


<PGROUP>
<PERSONA>MARK ANTONY</PERSONA>
<PERSONA>OCTAVIUS CAESAR</PERSONA>
<PERSONA>M. AEMILIUS LEPIDUS</PERSONA>
<GRPDESCR>triumvirs.</GRPDESCR>
</PGROUP>

<PERSONA>SEXTUS POMPEIUS</PERSONA>

<PGROUP>
<PERSONA>DOMITIUS ENOBARBUS</PERSONA>
<PERSONA>VENTIDIUS</PERSONA>
<PERSONA>EROS</PERSONA>
<PERSONA>SCARUS</PERSONA>
<PERSONA>DERCETAS</PERSONA>
<PERSONA>DEMETRIUS</PERSONA>
<PERSONA>PHILO</PERSONA>
<GRPDESCR>friends to Antony.</GRPDESCR>
</PGROUP>


<PGROUP>
<PERSONA>MECAENAS</PERSONA>
<PERSONA>AGRIPPA</PERSONA>
<PERSONA>DOLABELLA</PERSONA>
<PERSONA>PROCULEIUS</PERSONA>
<PERSONA>THYREUS</PERSONA>
<PERSONA>GALLUS</PERSONA>
<PERSONA>MENAS</PERSONA>
<GRPDESCR>friends to Caesar.</GRPDESCR>
</PGROUP>
...
...
...
```

```
<SPEECH>
<SPEAKER>First Guard</SPEAKER>
<LINE>This is an aspic's trail: and these fig-leaves</LINE>
<LINE>Have slime upon them, such as the aspic leaves</LINE>
<LINE>Upon the caves of Nile.</LINE>
</SPEECH>

<SPEECH>
<SPEAKER>OCTAVIUS CAESAR</SPEAKER>
<LINE>Most probable</LINE>
<LINE>That so she died; for her physician tells me</LINE>
<LINE>She hath pursued conclusions infinite</LINE>
<LINE>Of easy ways to die. Take up her bed;</LINE>
<LINE>And bear her women from the monument:</LINE>
<LINE>She shall be buried by her Antony:</LINE>
<LINE>No grave upon the earth shall clip in it</LINE>
<LINE>A pair so famous. High events as these</LINE>
<LINE>Strike those that make them; and their story is</LINE>
<LINE>No less in pity than his glory which</LINE>
<LINE>Brought them to be lamented. Our army shall</LINE>
<LINE>In solemn show attend this funeral;</LINE>
<LINE>And then to Rome. Come, Dolabella, see</LINE>
<LINE>High order in this great solemnity.</LINE>
</SPEECH>

<STAGEDIR>Exeunt</STAGEDIR>
</SCENE>
</ACT>
</PLAY>
```

XML Parser for C++ Example 3: XSL — iden.xsl

This example stylesheet can be used to input `XSLSample.cpp`.

```
<?xml version="1.0"?>
<!-- Identity transformation -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="*|@*|comment()|processing-instruction()|text()">
      <xsl:copy>
          <xsl:apply-templates
select="*|@*|comment()|processing-instruction()|text()"/>
      </xsl:copy>
  </xsl:template>
```

```
</xsl:stylesheet>
```

XML Parser for C++ Example 4: XML — FullDOM.xml (DTD)

This example DTD file inputs `FullDOM.cpp`.

```
<!DOCTYPE doc [
    <!ELEMENT p (#PCDATA)>
    <!ATTLIST p xml:space (preserve|default) 'preserve'>
    <!NOTATION notation1 SYSTEM "file.txt">
    <!NOTATION notation2 PUBLIC "some notation">
    <!ELEMENT doc (p*)>
    <!ENTITY example "<p>An ampersand (&#38;#38;) may be escaped
numerically (&#38;#38;#38;) or with a general entity
(&amp;amp;).</p>">
]>
<doc xml:lang="foo">&example;</doc>
```

XML Parser for C++ Example 5: XML — NSExample.xml

The following example file, `NSExample.xml`, uses namespaces.

```
<!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>
```

XML Parser for C++ Example 6: C++ — DOMSample.cpp

This example contains the C++ source code for DOMSample.cpp.

```
// Copyright (c) Oracle Corporation 1999, 2000. All Rights Reserved.
```

```
////////////////////////////////////////////////////////////////////////
// NAME
//   DOMSample.cpp
//
// DESCRIPTION
//   Sample usage of C++ XML parser via DOM interface
//
// PUBLIC FUNCTION(S)
//
// PRIVATE FUNCTION(S)
//
// NOTES
//   none
////////////////////////////////////////////////////////////////////////

#include <iostream.h>
#include <string.h>

#ifndef ORAXMLDOM_ORACLE
# include <oraxmldom.h>
#endif

#define DOCUMENT"cleo.xml"
#define DEFAULT_SPEAKER"Soothsayer"

void dump(Node *node);
void dumpspeech(Node *node);

char *speaker;
char *act, *scene;
uword n_speech;

int main(int argc, char **argv)
{
    XMLParser   parser;
    ub4         flags;
    uword       ecode;
    flags = XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE;

    cout << "XML C++ DOM sample\n";

    speaker = (argc > 1) ? argv[1] : DEFAULT_SPEAKER;

    cout << "Initializing XML package...\n";
```

```
    if (ecode = parser.xmlinit())
    {
        cout << "Failed to initialize XML parser, error " << ecode;
        return 1;
    }

    cout << "Parsing '" << DOCUMENT << "'...\n";
    cout.flush();
    if (ecode = parser.xmlparse((oratext *) DOCUMENT, (oratext *) 0, flags))
return 1;

    cout << "Dumping " << speaker << " speeches...\n";
    cout.flush();
    cout << "-------------------------------------------------------------\n";
    act = scene = "";
    n_speech = 0;
    dump(parser.getDocumentElement());

    (void) parser.xmlterm();// terminate LPX package

    return 0;
}

void dump(Node *node)
{
    Node *title, *speak;
    char *name, *who;
    uword i, n_nodes;

    name = (char *) node->getName();
    if (!strcmp((char *) name, "ACT"))
    {
title = node->getFirstChild();
act = (char *) title->getFirstChild()->getValue();
    }
    else if (!strcmp((char *) name, "SCENE"))
    {
title = node->getFirstChild();
scene = (char *) title->getFirstChild()->getValue();
    }
    else if (!strcmp((char *) name, "SPEECH"))
    {
speak = node->getFirstChild();
who = (char *) speak->getFirstChild()->getValue();
if (!strcmp(who, speaker))
```

```
    dumpspeech(node);
    }

    if (node->hasChildNodes())
    {
n_nodes = node->numChildNodes();
for (i = 0; i < n_nodes; i++)
    dump(node->getChildNode(i));
    }
}

// <SPEECH>
// <SPEAKER>Soothsayer</SPEAKER>
// <LINE>Your will?</LINE>
// </SPEECH>

// <SPEECH>
// <SPEAKER>CLEOPATRA</SPEAKER>
// <LINE><STAGEDIR>Aside to DOMITIUS ENOBARBUS</STAGEDIR>  What means
this?</LINE>
// </SPEECH>

void dumpspeech(Node *node)
{
    Node    *kid, *part, *partkid;
    uword    i, j, n_node, n_part;
    oratext *partname, *partval;

    if (n_speech++)
cout << "\n";
    cout << act << ", " << scene << "\n";
    n_node = node->numChildNodes();
    for (i = 0; i < n_node; i++)// skip speaker
    {
kid = node->getChildNode(i);// line #i
if (!strcmp((char *) kid->getName(), "LINE"))
{
    n_part = kid->numChildNodes();
    for (j = 0; j < n_part; j++)
    {
part = kid->getChildNode(j);
if (part->getType() == TEXT_NODE)
    cout << "    " << (char *) part->getValue() << "\n";
else
{
```

```
    partname = part->getName();
    partval = part->getFirstChild()->getValue();
    if (!strcmp((char *) partname, "STAGEDIR"))
cout << "    [" << (char *) partval << "]\n";
    else
cout << "    {" << (char *) partval << "}\n";
}
    }
}
    }
    cout.flush();
}

// end of DOMSample.c
```

XML Parser for C++ Example 7: C++ — DOMSample.std

`DOMSample.std` shows the expected output from `DOMSample.cpp`.

```
XML C++ DOM sample
Initializing XML package...
Parsing 'cleo.xml'...
Dumping Soothsayer speeches...
-------------------------------------------------------
ACT I, SCENE II.  The same. Another room.
    Your will?

ACT I, SCENE II.  The same. Another room.
    In nature's infinite book of secrecy
    A little I can read.

ACT I, SCENE II.  The same. Another room.
    I make not, but foresee.

ACT I, SCENE II.  The same. Another room.
    You shall be yet far fairer than you are.

ACT I, SCENE II.  The same. Another room.
    You shall be more beloving than beloved.

ACT I, SCENE II.  The same. Another room.
    You shall outlive the lady whom you serve.

ACT I, SCENE II.  The same. Another room.
```

```
    You have seen and proved a fairer former fortune
    Than that which is to approach.

ACT I, SCENE II.  The same. Another room.
    If every of your wishes had a womb.
    And fertile every wish, a million.

ACT I, SCENE II.  The same. Another room.
    Your fortunes are alike.

ACT I, SCENE II.  The same. Another room.
    I have said.

ACT II, SCENE III.  The same. OCTAVIUS CAESAR's house.
    Would I had never come from thence, nor you Thither!

ACT II, SCENE III.  The same. OCTAVIUS CAESAR's house.
    I see it in
    My motion, have it not in my tongue: but yet
    Hie you to Egypt again.

ACT II, SCENE III.  The same. OCTAVIUS CAESAR's house.
    Caesar's.
    Therefore, O Antony, stay not by his side:
    Thy demon, that's thy spirit which keeps thee, is
    Noble, courageous high, unmatchable,
    Where Caesar's is not; but, near him, thy angel
    Becomes a fear, as being o'erpower'd: therefore
    Make space enough between you.

ACT II, SCENE III.  The same. OCTAVIUS CAESAR's house.
    To none but thee; no more, but when to thee.
    If thou dost play with him at any game,
    Thou art sure to lose; and, of that natural luck,
    He beats thee 'gainst the odds: thy lustre thickens,
    When he shines by: I say again, thy spirit
    Is all afraid to govern thee near him;
    But, he away, 'tis noble.
```

XML Parser for C++ Example 8: C++ — SAXSample.cpp

This example contains the C++ source code for `SAXSample.cpp`.

```
// Copyright (c) Oracle Corporation 1999, 2000. All Rights Reserved.
```

```
////////////////////////////////////////////////////////////////////////
// NAME
//   SAXSample.cpp
//
// DESCRIPTION
//   Sample usage of C++ XML parser via SAX interface
//
// PUBLIC FUNCTION(S)
//
// PRIVATE FUNCTION(S)
//
// NOTES
//   none
////////////////////////////////////////////////////////////////////////

#include <iostream.h>
#include <string.h>

#ifndef ORAXMLDOM_ORACLE
# include <oraxmldom.h>
#endif

#define DOCUMENT"cleo.xml"
#define MAX_STRING128
#define MAX_SPEAKER20

oratext  elem[MAX_STRING], last_elem[MAX_STRING];
uword    n_speaker;
oratext *speakers[MAX_SPEAKER];
size_t   speakerlen[MAX_SPEAKER];

/* SAX callback functions */

sword startDocument(void *ctx);
sword endDocument(void *ctx);
sword startElement(void *ctx, const oratext *name,
   const struct xmlnodes *attrs);
sword endElement(void *ctx, const oratext *name);
sword characters(void *ctx, const oratext *ch, size_t len);

xmlsaxcb saxcb = {
    startDocument,
    endDocument,
    startElement,
```

```
    endElement,
    characters
};

int main()
{
    XMLParser   parser;
    ub4         flags;
    uword       ecode;
    flags = XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE;

    cout << "XML C++ SAX sample\n";

    cout << "Initializing XML package...\n";

    if (ecode = parser.xmlinit((oratext *) 0,
// encoding
  (void (*)(void *, const oratext *, ub4)) 0,
  (void *) 0,
// msghdlr ctx
  (xmlsaxcb *) &saxcb))
// SAX callback
    {
    cout << "Failed to initialize XML parser, error " << ecode;
    return 1;
    }

    cout << "Parsing '" << DOCUMENT << "' and showing speakers by scene...\n";
    cout.flush();
    if (ecode = parser.xmlparse((oratext *) DOCUMENT, (oratext *) 0, flags))
return 1;

    (void) parser.xmlterm();
// terminate LPX package

    return 0;
}

sword startDocument(void *ctx)
{
    cout << "startDocument\n";
    return 0;
}

sword endDocument(void *ctx)
```

```
{
    cout << "endDocument\n";
    return 0;
}

sword startElement(void *ctx, const oratext *name,
   const struct xmlnodes *attrs)
{
    strcpy((char *) last_elem, (char *) elem);
    strcpy((char *) elem, (char *) name);
    return 0;
}

sword endElement(void *ctx, const oratext *name)
{
    uword i;

    if (!strcmp((char *) name, "SCENE"))
    {
for (i = 0; i < n_speaker; i++)
{
    cout << "    ";
    cout.write(speakers[i], speakerlen[i]);
    cout << "\n";
}
    }
    return 0;
}

sword characters(void *ctx, const oratext *ch, size_t len)
{
    uword i;

    if (!strcmp((char *) elem, "TITLE"))
    {
if (!strcmp((char *) last_elem, "ACT"))
{
    cout << "\n--- ";
    cout.write(ch, len);
    cout << " ---\n\n";
}
else if (!strcmp((char *) last_elem, "SCENE"))
{
    n_speaker = 0;
    cout << "  ";
```

```
    cout.write(ch, len);
    cout << "\n";
}
    }
    else if (!strcmp((char *) elem, "SPEAKER"))
    {
if (n_speaker < MAX_SPEAKER)
{
    for (i = 0; i < n_speaker; i++)
if ((len == speakerlen[i]) && !strncmp((char *) speakers[i],
(char *) ch, len))
    break;
    if (!n_speaker || (i == n_speaker))
    {
speakers[n_speaker] = (oratext *) ch;
speakerlen[n_speaker++] = len;
    }
}
    }
    return 0;
}

// end of SAXSample.cc
```

XML Parser for C++ Example 9: C++ — SAXSample.std

`SAXSample.std` shows the expected output from `SAXSample.cpp`.

```
XML C++ SAX sample
Initializing XML package...
Parsing 'cleo.xml' and showing speakers by scene...
startDocument

--- ACT I ---

  SCENE I.  Alexandria. A room in CLEOPATRA's palace.
    PHILO
    CLEOPATRA
    MARK ANTONY
    Attendant
    DEMETRIUS
  SCENE II.  The same. Another room.
    CHARMIAN
    ALEXAS
```

```
    Soothsayer
    DOMITIUS ENOBARBUS
    IRAS
    CLEOPATRA
    Messenger
    MARK ANTONY
    First Attendant
    Second Attendant
    Second Messenger
  SCENE III.  The same. Another room.
    CLEOPATRA
    CHARMIAN
    MARK ANTONY
  SCENE IV.  Rome. OCTAVIUS CAESAR's house.
    OCTAVIUS CAESAR
    LEPIDUS
    Messenger
  SCENE V.  Alexandria. CLEOPATRA's palace.
    CLEOPATRA
    CHARMIAN
    MARDIAN
    ALEXAS

--- ACT II ---

...
...
--- ACT V ---
  SCENE I.  Alexandria. OCTAVIUS CAESAR's camp.
    OCTAVIUS CAESAR
    DOLABELLA
    DERCETAS
    AGRIPPA
    MECAENAS
    Egyptian
    PROCULEIUS
    All
  SCENE II.  Alexandria. A room in the monument.
    CLEOPATRA
    PROCULEIUS
    GALLUS
    IRAS
    CHARMIAN
    DOLABELLA
    OCTAVIUS CAESAR
```

```
    SELEUCUS
    Guard
    Clown
    First Guard
    Second Guard
endDocument
```

XML Parser for C++ Example 10: C++ — DOMNamespace.cpp

This example contains the C++ source code for DOMNamespace.cpp.

```
// Copyright (c) Oracle Corporation 1999, 2000. All Rights Reserved.

/////////////////////////////////////////////////////////////////////////
// NAME
//   DOMNamespace.cpp
//
// DESCRIPTION
//   This file demonstates a simple use of the parser and Namespace
//   extensions to the DOM APIs.
//
//   The XML file that is given to the application is parsed and the
//   elements and attributes in the document are printed.
//
// PUBLIC FUNCTION(S)
//
// PRIVATE FUNCTION(S)
//
// NOTES
//   none
/////////////////////////////////////////////////////////////////////////

#include <iostream.h>

#ifndef ORAXMLDOM_ORACLE
# include <oraxmldom.h>
#endif

#define DOCUMENT         "NSExample.xml"

void dump(Node *node);
void dumpattrs(Node *node);

//
```

```
// main
//

int main()
{
    XMLParser   parser;
    ub4         flags;
    uword       ecode;
    flags = XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE;

    cout << "\nXML C++ DOM Namespace\n";
    cout << "Initializing XML package...\n";
    if (ecode = parser.xmlinit())
    {
        cout << "Failed to initialize XML parser, error " << ecode;
        return 1;
    }

    cout << "Parsing '" << DOCUMENT << "'...\n";
    cout.flush();
    if (ecode = parser.xmlparse((oratext *) DOCUMENT, (oratext *) 0, flags))
return 1;

    cout << "\nThe elements are:\n";
    dump(parser.getDocumentElement());
    (void) parser.xmlterm();// terminate LPX package
    return 0;
}

//
// dump
//

void dump(Node *node)
{
    uword i, n_nodes;
    NodeList *nodes;
    size_t   nn;

    String qName;
    String localName;
    String nsName;
    String prefix;

    if (node == NULL)
```

```
      return;
   if (nodes = node->getChildNodes())
   {
      for (nn = node->numChildNodes(), i=0; i < nn; i++)
      {
         // Use the methods getQualifiedName(), getLocalName(),
         // getPrefix(), and getNamespace() to get Namespace
         // information.

         qName = prefix = localName = nsName = (oratext *)" ";

  if (node->getQualifiedName() != (oratext *)NULL)
            qName = node->getQualifiedName();

  if (node->getPrefix() != (oratext *)NULL)
            prefix = node->getPrefix();

  if (node->getLocal() != (oratext *)NULL)
            localName = node->getLocal();

  if (node->getNamespace() != (oratext *)NULL)
            nsName = node->getNamespace();
         cout << "  ELEMENT Qualified Name: " << (char *)qName << "\n";
         cout << "  ELEMENT Prefix        : " << (char *)prefix << "\n";
         cout << "  ELEMENT Local Name    : " << (char *)localName << "\n";
         cout << "  ELEMENT Namespace     : " << (char *)nsName << "\n";
         dumpattrs(node);
         dump(node->getChildNode(i));
      }
   }
}

//
// dumpattrs
//

void dumpattrs(Node *node)
{
    NamedNodeMap  *attrs;
    Attr          *a;
    uword          i;
    size_t         na;

    oratext   *qname;
    oratext   *namespce;
```

```
    oratext   *local;
    oratext   *prefix;
    oratext   *value;
    if (attrs = node->getAttributes())
    {
       cout << "\n    ATTRIBUTES: \n";
       for (na = attrs->getLength(), i = 0; i < na; i++)
       {
          /* get attr qualified name, local name, namespace, and prefix */

          a = (Attr *)attrs->item(i);
          qname = namespce = local = prefix = value = (oratext*)" ";
          if (a->getQualifiedName() != (oratext*)NULL)
             qname = a->getQualifiedName();
          if (a->getNamespace() != (oratext*)NULL)
             namespce = a->getNamespace();
          if (a->getLocal() != (oratext*)NULL)
             local = a->getLocal();
          if (a->getPrefix() != (oratext*)NULL)
             prefix = a->getPrefix();
          if (a->getValue() != (oratext*)NULL)
             value = a->getValue();

          cout << "      " << (char*)qname << " = " << (char*)value << "\n";
          cout << "      Namespace : " << (char*)namespce << "\n";
          cout << "      Local Name: " << (char*)local << "\n";
          cout << "      Prefix    : " << (char*)prefix << "\n\n";
       }
    }
    cout << "\n";
}
```

XML Parser for C++ Example 11: C++ — DOMNamespace.std

DOMNamespace.std shows the expected output from `DOMNamespace.cpp`.

```
XML C++ DOM Namespace
Initializing XML package...
Parsing 'NSExample.xml'...

The elements are:
  ELEMENT Qualified Name: doc
  ELEMENT Prefix        :
  ELEMENT Local Name    : doc
```

```
ELEMENT Namespace     : http://www.w3c.org

  ATTRIBUTES:
    nsprefix:a1 = v1
    Namespace : http://www.oracle.com
    Local Name: a1
    Prefix    : nsprefix

    xmlns = http://www.w3c.org
    Namespace :
    Local Name: xmlns
    Prefix    :

    xmlns:nsprefix = http://www.oracle.com
    Namespace :
    Local Name: nsprefix
    Prefix    : xmlns

ELEMENT Qualified Name: child
ELEMENT Prefix        :
ELEMENT Local Name    : child
ELEMENT Namespace     : http://www.w3c.org
```

XML Parser for C++ Example 12: C++ — SAXNamespace.cpp

This example contains the C++ source code for the SAXNamespace program.

```
// Copyright (c) Oracle Corporation 1999, 2000. All Rights Reserved.

///////////////////////////////////////////////////////////////////////////
// NAME
//   DOMNamespace.cpp
//
// DESCRIPTION
//   This file demonstates a simple use of the parser and Namespace
//   extensions to the SAX APIs.
//   The XML file that is given to the application is parsed and the
//   elements and attributes in the document are printed.
//
// PUBLIC FUNCTION(S)
//
// PRIVATE FUNCTION(S)
//
// NOTES
```

```
//   none
////////////////////////////////////////////////////////////////////////

#include <iostream.h>

#ifndef ORAXMLDOM_ORACLE
# include <oraxmldom.h>
#endif

#define DOCUMENT         "NSExample.xml"

/*---------------------------------------------------------------------
                          FUNCTION PROTOTYPES
  ---------------------------------------------------------------------*/
int startDocument(void *ctx);
int endDocument(void *ctx);
int endElement(void *ctx, const oratext *name);
int nsStartElement(void *ctx, const oratext *qname,
                       const oratext *local,
                       const oratext *nsp,
                       const struct xmlnodes *attrs);

/* SAX callback structure */

xmlsaxcb saxcb = {
    startDocument,
    endDocument,
    0,
    endElement,
    0,
    0,
    0,
    0,
    0,
    nsStartElement,
    0, 0, 0, 0, 0, 0, 0, 0
};

/* SAX callback context */
/*
typedef struct {
    xmlctx  *ctx;
    uword    depth;
} cbctx;
*/
```

```
/*---------------------------------------------------------------------
                              MAIN
  ---------------------------------------------------------------------*/
int main()
{
   XMLParser   parser;
   ub4         flags;
   uword       ecode;
   flags = XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE;
   cout << "XML C++ SAX Namespace\n";
   cout << "Initializing XML package...\n";
   if (ecode = parser.xmlinit((oratext *) 0,           // encoding
                              (void (*)(void *, const oratext *, ub4)) 0,
                              (void *) 0,           // msghdlr ctx
                              (xmlsaxcb *) &saxcb)) // SAX callback
   {
       cout << "Failed to initialize XML parser, error " << ecode;
       return 1;
   }

   /* parse the document */

   cout << "Parsing '" << DOCUMENT << "'...\n";
   cout.flush();
   if (ecode = parser.xmlparse((oratext *) DOCUMENT, (oratext *) 0, flags))
      return 1;

   (void) parser.xmlterm();// terminate LPX package

   return 0;
}

/*---------------------------------------------------------------------
                            SAX Interface
  ---------------------------------------------------------------------*/
int startDocument(void *ctx)
{
    cout << "\nStartDocument\n\n";
    return 0;
}

int endDocument(void *ctx)
{
    cout << "\nEndDocument\n";
```

```
    return 0;
}

int endElement(void *ctx, const oratext *name)
{
    cout << "\nELEMENT Name  : " << (char*)name << "\n";
    return 0;
}

int nsStartElement(void *ctx, const oratext *qname, const oratext *local,
                   const oratext *nsp, const struct xmlnodes *attrs)
{
    xmlnode *attr;
    uword    i;
    oratext *aqname;
    oratext *alocal;
    oratext *anamespace;
    oratext *aprefix;
    oratext *avalue;

    /*
     * Use the functions getXXXQualifiedName(), getXXXLocalName(), and
     * getXXXNamespace() to get Namespace information.
     */

    if (qname == (oratext*)NULL)
       qname = (oratext*)" ";
    if (local == (oratext*)NULL)
       local = (oratext*)" ";
    if (nsp == (oratext*)NULL)
       nsp = (oratext*)" ";

    cout << "ELEMENT Qualified Name: " << (char*)qname << "\n";
    cout << "ELEMENT Local Name    : " << (char*)local << "\n";
    cout << "ELEMENT Namespace     : " << (char*)nsp << "\n";

    if (attrs)
    {
       for (i = 0; i < numAttributes(attrs); i++)
       {
          attr = getAttributeIndex(attrs,i);
          aqname = alocal = anamespace = aprefix = avalue = (oratext*)" ";

          if (getAttrQualifiedName(attr))
             aqname = (oratext *) getAttrQualifiedName(attr);
```

```
            if (getAttrPrefix(attr))
               aprefix = (oratext *) getAttrPrefix(attr);
            if (getAttrLocal(attr))
               alocal = (oratext *) getAttrLocal(attr);
            if (getAttrNamespace(attr))
               anamespace = (oratext *) getAttrNamespace(attr);
            if (getAttrValue(attr))
               avalue = (oratext *) getAttrValue(attr);

           cout << " ATTRIBUTE Qualified Name   : " << (char*)aqname << "\n";
           cout << " ATTRIBUTE Prefix           : " << (char*)aprefix << "\n";
           cout << " ATTRIBUTE Local Name       : " << (char*)alocal << "\n";
           cout << " ATTRIBUTE Namespace        : " << (char*)anamespace << "\n";
           cout << " ATTRIBUTE Value            : " << (char*)avalue << "\n";
           cout << "\n";
         }
      }
      return 0;
}
```

XML Parser for C++ Example 13: C++ — SAXNamespace.std

`SAXNamespace.std` shows the expected output from `SAXNamespace.cpp`.

```
XML C++ SAX Namespace
Initializing XML package...
Parsing 'NSExample.xml'...

StartDocument

ELEMENT Qualified Name: doc
ELEMENT Local Name    : doc
ELEMENT Namespace     : http://www.w3c.org
 ATTRIBUTE Qualified Name   : nsprefix:a1
 ATTRIBUTE Prefix           : nsprefix
 ATTRIBUTE Local Name       : a1
 ATTRIBUTE Namespace        : http://www.oracle.com
 ATTRIBUTE Value            : v1

 ATTRIBUTE Qualified Name   : xmlns
 ATTRIBUTE Prefix           :
 ATTRIBUTE Local Name       : xmlns
 ATTRIBUTE Namespace        :
 ATTRIBUTE Value            : http://www.w3c.org
```

```
 ATTRIBUTE Qualified Name   : xmlns:nsprefix
 ATTRIBUTE Prefix           : xmlns
 ATTRIBUTE Local Name       : nsprefix
 ATTRIBUTE Namespace        :
 ATTRIBUTE Value            : http://www.oracle.com

ELEMENT Qualified Name: child
ELEMENT Local Name    : child
ELEMENT Namespace     : http://www.w3c.org

ELEMENT Name  : child
ELEMENT Name  : doc
EndDocument
```

XML Parser for C++ Example 14: C++ — FullDOM.cpp

This example contains the C++ source code for `FullDOM.cpp`.

```
// Copyright (c) Oracle Corporation 1999, 2000. All Rights Reserved.

///////////////////////////////////////////////////////////////////////////
// NAME
//   FullDOM.cpp
//
// DESCRIPTION
//   Sample code to test full C++ DOM interface
///////////////////////////////////////////////////////////////////////////

#include <iostream.h>

#ifndef ORAXMLDOM_ORACLE
# include <oraxmldom.h>
#endif

#define TEST_DOC(oratext *) "FullDOM.xml"

void dump(Node *node, uword level);
void dumpnode(Node *node, uword level);

static char *ntypename[] = {
    "0",
    "ELEMENT",
    "ATTRIBUTE",
```

```
    "TEXT",
    "CDATA",
    "ENTREF",
    "ENTITY",
    "PI",
    "COMMENT",
    "DOCUMENT",
    "DTD",
    "DOCFRAG",
    "NOTATION"
};

#define FAIL { cout << "Failed!\n"; return 1; }

int main()
{
    XMLParser     parser;
    Document     *doc;
    Element      *root, *elem, *subelem;
    Attr         *attr, *attr1, *attr2, *gleep1, *gleep2;
    Text         *text, *subtext;
    Node         *node, *pi, *comment, *entref, *cdata, *clone,
 *deep_clone, *frag, *fragelem, *fragtext, *sub2,
 *fish, *food, *food2, *repl;
    NodeList     *subs, *nodes;
    NamedNodeMap *attrs, *notes, *entities;
    DocumentType *dtd;
    uword         i, ecode, level;

    cout << "XML C++ Full DOM test\n";
    cout << "Initializing XML parser...\n";

    if (ecode = parser.xmlinit())
    {
cout << "Failed to initialze XML parser, error " << ecode << "\n";
return 1;
    }

    cout << "\nCreating new document...\n";
    if (!(doc = parser.createDocument()))
FAIL

    cout << "Document from root node:\n";
    dump(parser.getDocument(), 0);
```

```
    cout << "\nCreating root element ('ROOT')...\n";
    if (!(elem = doc->createElement((oratext *) "ROOT")))
FAIL

    cout << "Setting as root element...\n";
    if (!doc->appendChild(elem))
FAIL

    cout << "Document from 'ROOT' element:\n";
    dump(root = parser.getDocumentElement(), 0);
    cout << "Adding 7 children to 'ROOT' element...\n";
    if (!(text = doc->createTextNode((oratext *) "Gibberish")) ||
        !elem->appendChild(text))
FAIL
    if (!(comment = doc->createComment((oratext*) "Bit warm today, innit?")) ||
        !elem->appendChild(comment))
FAIL

    if (!(pi = doc->createProcessingInstruction((oratext *) "target",
(oratext *) "PI-contents")) ||
       !elem->appendChild(pi))
FAIL

    if (!(cdata = doc->createCDATASection((oratext *) "See DATA")) ||
        !elem->appendChild(cdata))
FAIL

    if (!(entref = doc->createEntityReference((oratext *) "EntRef")) ||
        !elem->appendChild(entref))
FAIL

    if (!(fish = doc->createElement((oratext *) "FISH")) ||
!elem->appendChild(fish))
FAIL

    if (!(food = doc->createElement((oratext *) "FOOD")) ||
!elem->appendChild(food))
FAIL
    cout << "Document from 'ROOT' element with its 7 children:\n";
    dump(root, 0);

    cout << "\nTesting node insertion...\n";
    cout << "Adding 'Pre-Gibberish' text node and 'Ask about the weather'
comment node...\n";
    if (!(node = doc->createTextNode((oratext *) "Pre-Gibberish")) ||
```

```
        !elem->insertBefore(node, text))
FAIL

    if (!(node = doc->createComment((oratext *) "Ask about the weather:")) ||
        !elem->insertBefore(node, comment))
FAIL

    cout << "Document from 'ROOT' element:\n";
    dump(root, 0);
    cout << "Document from 'ROOT' element:\n";
    dump(root, 0);
    cout << "Document from 'ROOT' element:\n";
    dump(root, 0);
    cout << "\nTesting nextSibling links starting at first child...\n";
    for (node = elem->getFirstChild();
    node;
    node = node->getNextSibling())dump(node, 1);
    cout << "\nTesting previousSibling links starting at last child...\n";
    for (node = elem->getLastChild();
    node;
    node = node->getPreviousSibling())dump(node, 1);

    cout << "\nTesting setting node value...\n";
    cout << "Original node:\n";
    dump(pi, 1);
    pi->setValue((oratext *) "New PI contents");
    cout << "Node after new value:\n";
    dump(pi, 1);

    cout << "\nAdding another element level, i.e., 'SUB'...\n";
    if (!(subelem = doc->createElement((oratext *) "SUB")) ||
!elem->insertBefore(subelem, cdata) ||
!(subtext = doc->createTextNode((oratext *) "Lengthy SubText")) ||
        !subelem->appendChild(subtext))
FAIL

    cout << "Document from 'ROOT' element:\n";
    dump(root, 0);

    cout << "\nAdding a second 'SUB' element...\n";
    if (!(sub2 = doc->createElement((oratext *) "SUB")) ||
!elem->insertBefore(sub2, cdata))
FAIL

    cout << "Document from 'ROOT' element:\n";
```

```
    dump(root, 0);

    cout << "\nGetting all SUB nodes - note the distinct hex addresses...\n";
    if (!(subs = doc->getElementsByTagName(root, (oratext *) "SUB")))
FAIL
    for (i = 0; i < subs->getLength(); i++)
dumpnode(subs->item(i), 1);

    cout << "\nTesting parent links...\n";
    for (level = 1, node = subtext; node; node = node->getParentNode(), level++)
dumpnode(node, level);

    cout << "\nTesting owner document of node...\n";
    dumpnode(subtext, 1);
    dumpnode(subtext->getOwnerDocument(), 1);

    cout << "\nTesting node replacement...\n";
    if (!(node = doc->createTextNode((oratext *) "REPLACEMENT, 1/2 PRICE")) ||
        !pi->replaceChild(node))
FAIL

    cout << "Document from 'ROOT' element:\n";
    dump(root, 0);

    cout << "\nTesting node removal...\n";
    if (!entref->removeChild())
FAIL

    cout << "Document from 'ROOT' element:\n";
    dump(root, 0);
    cout << "\nNormalizing...\n";
    elem->normalize();
    cout << "Document from 'ROOT' element:\n";
    dump(root, 0);
    cout << "\nCreating and populating document fragment...\n";
    if (!(frag = doc->createDocumentFragment()) ||
!(fragelem = doc->createElement((oratext *) "FragElem")) ||
!(fragtext = doc->createTextNode((oratext *) "FragText")) ||
!frag->appendChild(fragelem) ||
!frag->appendChild(fragtext))
FAIL
    dump(frag, 1);
    cout << "Insert document fragment...\n";
    if (!elem->insertBefore(frag, comment))
FAIL
```

```
    dump(elem, 1);

    cout << "\nCreate two attributes...\n";
    if (!(attr1 = doc->createAttribute((oratext*)"Attr1",(oratext*)"Value1")) ||
!(attr2 = doc->createAttribute((oratext*)"Attr2",(oratext*)"Value2")))
FAIL
    cout << "Setting attributes...\n";
    if (!subelem->setAttributeNode(attr1, NULL) ||
!subelem->setAttributeNode(attr2, NULL))
FAIL
    dump(subelem, 1);

    cout << "\nAltering attribute1 value...\n";
    attr1->setValue((oratext *) "New1");
    dump(subelem, 1);

    cout << "\nFetching attribute by name (Attr2)...\n";
    if (!(node = subelem->getAttributeNode((oratext *) "Attr2")))
FAIL
    dump(node, 1);

    cout << "\nRemoving attribute by name (Attr1)...\n";
    subelem->removeAttribute((oratext *) "Attr1");
    dump(subelem, 1);

    cout << "\nAdding new attribute...\n";
    if (!subelem->setAttribute((oratext *) "Attr3", (oratext *) "Value3"))
FAIL
    dump(subelem, 1);

    cout << "\nRemoving attribute by pointer (Attr2)...\n";
    if (!subelem->removeAttributeNode(attr2))
FAIL
    dump(subelem, 1);

    cout << "\nAdding new attribute w/same name (test replacement)...\n";
    dump(subelem, 1);
    if (!(attr = doc->createAttribute((oratext*)"Attr3", (oratext*)"Zoo3")))
FAIL
    if (!subelem->setAttributeNode(attr, NULL))
FAIL
    dump(subelem, 1);

    cout << "\nTesting node (attr) set by name...\n";
    cout << "Adding 'GLEEP' attr and printing out hex addresses of node set\n";
```

```
    attrs = subelem->getAttributes();
    if (!(gleep1=doc->createAttribute((oratext*)"GLEEP",(oratext*)"GLEEP1")) ||
!attrs->setNamedItem(gleep1, NULL))
FAIL
    dump(subelem, 0);

    cout << "\nTesting node (attr) set by name...\n";
    cout << "Replacing 'GLEEP' element - note the changed hex address\n";
    if (!(gleep2=doc->createAttribute((oratext*)"GLEEP",(oratext*)"GLEEP2")) ||
!attrs->setNamedItem(gleep2, &repl))
FAIL
    dump(subelem, 0);
    cout << "Replaced node was:\n";
    dump(repl, 1);

    cout << "\nTesting node removal by name...\n";
    cout << "Removing 'GLEEP' attribute\n";
    if (!attrs->removeNamedItem((oratext *) "GLEEP"))
FAIL
    dump(subelem, 0);

    cout << "\nOriginal SubROOT...\n";
    dump(subelem, 1);
    cout << "Cloned SubROOT (not deep)...\n";
    clone = subelem->cloneNode(FALSE);
    dump(clone, 1);
    cout << "Cloned SubROOT (deep)...\n";
    deep_clone = subelem->cloneNode(TRUE);
    dump(deep_clone, 1);

    cout << "\nSplitting text...\n";
    dump(subelem, 1);
    subtext->splitText(3);
    dump(subelem, 1);

    cout << "\nTesting string operations...\n";
    cout << "    CharData = \"" << (char *) subtext->getData() << "\"\n";
    cout << "Setting new data...\n";
    subtext->setData((oratext *) "0123456789");
    cout << "    CharData = \"" << (char *) subtext->getData() << "\"\n";
    cout << "    CharLength = " << (int) subtext->getLength() << "\n";
    cout << "    Substring(0,5) = \"" <<
(char *) subtext->substringData(0, 5) << "\"\n";
    cout << "    Substring(8,2) = \"" <<
(char *) subtext->substringData(8, 2) << "\"\n";
```

```
    cout << "Appending data...\n";
    subtext->appendData((oratext *) "ABCDEF");
    cout << "    CharData = \"" << (char *) subtext->getData() << "\"\n";
    cout << "Inserting data...\n";
    subtext->insertData(10, (oratext *) "*foo*");
    cout << "    CharData = \"" << (char *) subtext->getData() << "\"\n";
    cout << "Deleting data...\n";
    subtext->deleteData(0, 10);
    cout << "    CharData = \"" << (char *) subtext->getData() << "\"\n";
    cout << "Replacing data...\n";
    subtext->replaceData(1, 3, (oratext *) "bamboozle");
    cout << "    CharData = \"" << (char *) subtext->getData() << "\"\n";

    cout << "Cleaning up...\n";
    parser.xmlclean();

    if (parser.getDocument())
    {
cout << "Problem, document is not gone!!\n";
return 1;
    }

    cout << "Parsing test document...\n";
    if (ecode = parser.xmlparse(TEST_DOC, (oratext *) 0, 0))
    {
cout << "Parse failed, code " << ecode << "\n";;
return ecode;
    }

    cout << "Document from root node:\n" << flush;
    dump(parser.getDocument(), 0);

    cout << "Testing getNotations...\n" << flush;
    dtd = parser.getDocType();
    if (notes = dtd->getNotations())
    {
cout << "# of notations = " << notes->getLength() << "\n" << flush;
for (i = 0; i < notes->getLength(); i++)
    dump(notes->item(i), 1);
    }
    else
cout << "No defined notations\n" << flush;

    cout << "Testing getEntities...\n" << flush;
    if (entities = dtd->getEntities())
```

```
    {
cout << "# of entities = " << entities->getLength() << "\n" << flush;
for (i = 0; i < entities->getLength(); i++)
    dump(entities->item(i), 1);
    }
    else
cout << "No defined entities\n" << flush;

    cout << "Cleaning up...\n";
    parser.xmlclean();

    if (parser.getDocument())
    {
cout << "Problem, document is not gone!!\n";
return 1;
    }

    cout << "\nTerminating parser...\n";
    parser.xmlterm();

    cout << "Success.\n";
    return 0;
}

void dump(Node *node, uword level)
{
    NodeList *nodes;
    uword     i, n_nodes;

    if (node)
    {
dumpnode(node, level);
if (node->hasChildNodes())
{
    nodes = node->getChildNodes();
    n_nodes = node->numChildNodes();
    for (i = 0; i < n_nodes; i++)
dump(nodes->item(i), level + 1);
}
    }
}

void dumpnode(Node *node, uword level)
{
    const oratext *name, *value;
```

```
    short          type;
    NamedNodeMap  *attrs;
    Attr          *attr;
    uword          i, n_attrs;

    if (node)
    {
for (i = 0; i <= level; i++)
    cout << "    ";
type = node->getType();
cout << (char *) ntypename[type];
if ((name = node->getName()) && (*name != '#'))
    cout << " \"" << (char *) name << "\"";
if (value = node->getValue())
    cout << " = \"" << (char *) value << "\"";
if ((type == ELEMENT_NODE) && (attrs = node->getAttributes()))
{
    cout << " [";
    n_attrs = attrs->getLength();
    for (i = 0; i < n_attrs; i++)
    {
if (i) cout << ", ";
attr = (Attr *) attrs->item(i);
cout << (char *) attr->getName();
if (attr->getSpecified())
    cout << "*";
cout << "=\"" << (char *) attr->getValue() << "\"";
    }
    cout << "]";
}
cout << "\n";
    }
}

// end of FullDOM.cpp
```

XML Parser for C++ Example 15: C++ — FullDOM.std

The FullDOM.std example file shows the expected output from FullDOM.cpp

```
XML C++ Full DOM test
Initializing XML parser...

Creating new document...
```

```
Document from root node:
    DOCUMENT

Creating root element ('ROOT')...
Setting as root element...
Document from 'ROOT' element:
    ELEMENT "ROOT"
Adding 7 children to 'ROOT' element...
Document from 'ROOT' element with its 7 children:
    ELEMENT "ROOT"
        TEXT = "Gibberish"
        COMMENT = "Bit warm today, innit?"
        PI "target" = "PI-contents"
        CDATA = "See DATA"
        ENTREF "EntRef"
        ELEMENT "FISH"
        ELEMENT "FOOD"

Testing node insertion...
Adding 'Pre-Gibberish' text node and 'Ask about the weather' comment node...
Document from 'ROOT' element:
    ELEMENT "ROOT"
        TEXT = "Pre-Gibberish"
        TEXT = "Gibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        PI "target" = "PI-contents"
        CDATA = "See DATA"
        ENTREF "EntRef"
        ELEMENT "FISH"
        ELEMENT "FOOD"
Document from 'ROOT' element:
    ELEMENT "ROOT"
        TEXT = "Pre-Gibberish"
        TEXT = "Gibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        PI "target" = "PI-contents"
        CDATA = "See DATA"
        ENTREF "EntRef"
        ELEMENT "FISH"
        ELEMENT "FOOD"
Document from 'ROOT' element:
    ELEMENT "ROOT"
        TEXT = "Pre-Gibberish"
```

```
        TEXT = "Gibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        PI "target" = "PI-contents"
        CDATA = "See DATA"
        ENTREF "EntRef"
        ELEMENT "FISH"
        ELEMENT "FOOD"

Testing nextSibling links starting at first child...
        TEXT = "Pre-Gibberish"
        TEXT = "Gibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        PI "target" = "PI-contents"
        CDATA = "See DATA"
        ENTREF "EntRef"
        ELEMENT "FISH"
        ELEMENT "FOOD"

Testing previousSibling links starting at last child...
        ELEMENT "FOOD"
        ELEMENT "FISH"
        ENTREF "EntRef"
        CDATA = "See DATA"
        PI "target" = "PI-contents"
        COMMENT = "Bit warm today, innit?"
        COMMENT = "Ask about the weather:"
        TEXT = "Gibberish"
        TEXT = "Pre-Gibberish"

Testing setting node value...
Original node:
        PI "target" = "PI-contents"
Node after new value:
        PI "target" = "New PI contents"

Adding another element level, i.e., 'SUB'...
Document from 'ROOT' element:
    ELEMENT "ROOT"
        TEXT = "Pre-Gibberish"
        TEXT = "Gibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        PI "target" = "New PI contents"
```

```
        ELEMENT "SUB"
            TEXT = "Lengthy SubText"
        CDATA = "See DATA"
        ENTREF "EntRef"
        ELEMENT "FISH"
        ELEMENT "FOOD"

Adding a second 'SUB' element...
Document from 'ROOT' element:
    ELEMENT "ROOT"
        TEXT = "Pre-Gibberish"
        TEXT = "Gibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        PI "target" = "New PI contents"
        ELEMENT "SUB"
            TEXT = "Lengthy SubText"
        ELEMENT "SUB"
        CDATA = "See DATA"
        ENTREF "EntRef"
        ELEMENT "FISH"
        ELEMENT "FOOD"

Getting all SUB nodes - note the distinct hex addresses...
        ELEMENT "SUB"
        ELEMENT "SUB"

Testing parent links...
        TEXT = "Lengthy SubText"
            ELEMENT "SUB"
                ELEMENT "ROOT"
                    DOCUMENT

Testing owner document of node...
        TEXT = "Lengthy SubText"
        DOCUMENT

Testing node replacement...
Document from 'ROOT' element:
    ELEMENT "ROOT"
        TEXT = "Pre-Gibberish"
        TEXT = "Gibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        TEXT = "REPLACEMENT, 1/2 PRICE"
```

```
        ELEMENT "SUB"
            TEXT = "Lengthy SubText"
        ELEMENT "SUB"
        CDATA = "See DATA"
        ENTREF "EntRef"
        ELEMENT "FISH"
        ELEMENT "FOOD"

Testing node removal...
Document from 'ROOT' element:
    ELEMENT "ROOT"
        TEXT = "Pre-Gibberish"
        TEXT = "Gibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        TEXT = "REPLACEMENT, 1/2 PRICE"
        ELEMENT "SUB"
            TEXT = "Lengthy SubText"
        ELEMENT "SUB"
        CDATA = "See DATA"
        ELEMENT "FISH"
        ELEMENT "FOOD"

Normalizing...
Document from 'ROOT' element:
    ELEMENT "ROOT"
        TEXT = "Pre-GibberishGibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        TEXT = "REPLACEMENT, 1/2 PRICE"
        ELEMENT "SUB"
            TEXT = "Lengthy SubText"
        ELEMENT "SUB"
        CDATA = "See DATA"
        ELEMENT "FISH"
        ELEMENT "FOOD"

Creating and populating document fragment...
        DOCFRAG
            ELEMENT "FragElem"
            TEXT = "FragText"
Insert document fragment...
        ELEMENT "ROOT"
            TEXT = "Pre-GibberishGibberish"
            COMMENT = "Ask about the weather:"
```

```
                ELEMENT "FragElem"
                TEXT = "FragText"
                COMMENT = "Bit warm today, innit?"
                TEXT = "REPLACEMENT, 1/2 PRICE"
                ELEMENT "SUB"
                    TEXT = "Lengthy SubText"
                ELEMENT "SUB"
                CDATA = "See DATA"
                ELEMENT "FISH"
                ELEMENT "FOOD"

Create two attributes...
Setting attributes...
        ELEMENT "SUB" [Attr1*="Value1", Attr2*="Value2"]
            TEXT = "Lengthy SubText"

Altering attribute1 value...
        ELEMENT "SUB" [Attr1*="New1", Attr2*="Value2"]
            TEXT = "Lengthy SubText"

Fetching attribute by name (Attr2)...
        ATTRIBUTE "Attr2" = "Value2"

Removing attribute by name (Attr1)...
        ELEMENT "SUB" [Attr2*="Value2"]
            TEXT = "Lengthy SubText"

Adding new attribute...
        ELEMENT "SUB" [Attr2*="Value2", Attr3*="Value3"]
            TEXT = "Lengthy SubText"

Removing attribute by pointer (Attr2)...
        ELEMENT "SUB" [Attr3*="Value3"]
            TEXT = "Lengthy SubText"

Adding new attribute w/same name (test replacement)...
        ELEMENT "SUB" [Attr3*="Value3"]
            TEXT = "Lengthy SubText"
        ELEMENT "SUB" [Attr3*="Zoo3"]
            TEXT = "Lengthy SubText"

Testing node (attr) set by name...
Adding 'GLEEP' attr and printing out hex addresses of node set
    ELEMENT "SUB" [Attr3*="Zoo3", GLEEP*="GLEEP1"]
        TEXT = "Lengthy SubText"
```

```
Testing node (attr) set by name...
Replacing 'GLEEP' element - note the changed hex address
    ELEMENT "SUB" [Attr3*="Zoo3", GLEEP*="GLEEP2"]
        TEXT = "Lengthy SubText"
Replaced node was:
        ATTRIBUTE "GLEEP" = "GLEEP1"

Testing node removal by name...
Removing 'GLEEP' attribute
    ELEMENT "SUB" [Attr3*="Zoo3"]
        TEXT = "Lengthy SubText"

Original SubROOT...
        ELEMENT "SUB" [Attr3*="Zoo3"]
            TEXT = "Lengthy SubText"
Cloned SubROOT (not deep)...
        ELEMENT "SUB" [Attr3*="Zoo3"]
            TEXT = "Lengthy SubText"
Cloned SubROOT (deep)...
        ELEMENT "SUB" [Attr3*="Zoo3"]
            TEXT = "Lengthy SubText"

Splitting text...
        ELEMENT "SUB" [Attr3*="Zoo3"]
            TEXT = "Lengthy SubText"
        ELEMENT "SUB" [Attr3*="Zoo3"]
            TEXT = "Leng"
            TEXT = "thy SubText"

Testing string operations...
    CharData = "Leng"
Setting new data...
    CharData = "0123456789"
    CharLength = 10
    Substring(0,5) = "01234"
    Substring(8,2) = "89"
Appending data...
    CharData = "0123456789ABCDEF"
Inserting data...
    CharData = "0123456789*foo*ABCDEF"
Deleting data...
    CharData = "*foo*ABCDEF"
Replacing data...
    CharData = "*bamboozle*ABCDEF"
```

```
Cleaning up...
Parsing test document...
Document from root node:
    DOCUMENT
        DTD "doc"
        ELEMENT "doc" [xml:lang*="foo"]
            ELEMENT "p" [xml:space="preserve"]
                TEXT = "An ampersand (&) may be escaped
numerically (&#38;) or with a general entity
(&amp;)."
Testing getNotations...
# of notations = 2
        NOTATION "notation1"
        NOTATION "notation2"
Testing getEntities...
# of entities = 1
        ENTITY "example" = "<p>An ampersand (&#38;) may be escaped
numerically (&#38;#38;) or with a general entity
(&amp;amp;).</p>"
Cleaning up...

Terminating parser...
Success.
```

XML Parser for C++ Example 16: C++ — XSLSample.cpp

This example contains the C++ source code for XSLSample.cpp

```
// Copyright (c) Oracle Corporation 1999. All Rights Reserved.

/////////////////////////////////////////////////////////////////////////////
// NAME
//   XSLSample.cpp
//
// DESCRIPTION
//   Sample usage of C++ XSL processor
//
// PUBLIC FUNCTION(S)
//
// PRIVATE FUNCTION(S)
//
// NOTES
//   none
/////////////////////////////////////////////////////////////////////////////
```

```
#ifndef ORAXMLDOM_ORACLE
# include <oraxmldom.h>
#endif

int main(int argc, char **argv)
{
    XMLParser     xmlpar, xslpar, respar;
    XSLProcessor  xslproc;
    Node         *result;
    ub4           flags;
    uword         ecode;
    flags = XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE;

    cout << "XSL processor sample\n";

    if (argc < 3)
    {
cout << "Usage is XSLSample <xmlfile> <stlyesheet>\n";
return 1;
    }

    // Parse the XML file
    cout << "Parsing XML file " << argv[1] << "\n";
    if (ecode = xmlpar.xmlinit())
    {
        cout << "Failed to initialize XML parser, error " << ecode << "\n";;
        return 1;
    }
    if (ecode = xmlpar.xmlparse((oratext *) argv[1], (oratext *) 0, flags))
        return 1;

    // Parse the Stylesheet file
    cout << "Parsing Stylesheet " << argv[2] << "\n";
    if (ecode = xslpar.xmlinit())
    {
        cout << "Failed to initialize XML parser, error " << ecode << "\n";;
        return 1;
    }
    if (ecode = xslpar.xmlparse((oratext *) argv[2], (oratext *) 0, flags))
        return 1;

    // Initialize the result context
    cout << "Initializing the result context\n";
    if (ecode = respar.xmlinit())
```

```
    {
        cout << "Failed to initialize XML parser, error " << ecode << "\n";;
        return 1;
    }

    // XSL Processing
    cout << "XSL Processing\n";
    if (ecode = xslproc.xslprocess(&xmlpar, &xslpar, &respar, &result))
    {
        cout << "Failed in XSL Processing, error " << ecode << "\n";;
        return 1;
    }

    // print the resultant tree
    cout.flush();
    xslproc.printres(&respar, result);


    // Terminate the parsers
    (void) xmlpar.xmlterm();
    (void) xslpar.xmlterm();
    (void) respar.xmlterm();

    return 0;
}
```

XML Parser for C++ Example 17: C++ — XSLSample.std

This example shows the typical result output from XSLSample.cpp

```
<xsl:param name="size"/>
<xsl:param name="data"/>
<xsl:choose><xsl:when test="number(number($size) <
string-length(string($data)))">
   <xsl:value-of select="substring(string($data), 1, number($size))"/>
</xsl:when>
<xsl:otherwise>
<xsl:value-of select="string($data)"/>
</xsl:otherwise></xsl:choose>
<xsl:if test="number(number($size) > string-length(string($data)))">
<xsl:call-template name="pad">
<xsl:with-param name="padsize" select="number($size) -
string-length(string($data))"/>
</xsl:call-template></xsl:if></xsl:template><xsl:template match="/">
```

```
<xsl:text>&#13;&#10;</xsl:text><xsl:apply-templates select="//ROWSE
T/ROW/CUSTOMER"/>
   <xsl:text>&#13;&#10;</xsl:text>
</xsl:template><xsl:template match="CUSTOMER">
<xsl:call-template name="truncateorpad">
<xsl:with-param name="size" select="31"/>
<xsl:with-param name="data" select ="."/>
</xsl:call-template>
</xsl:template>
</xsl:stylesheet>
```

23

Using XML C++ Class Generator

This chapter contains the following sections:

Accessing XML C++ Class Generator

The XML C++ Class Generator is provided with Oracle8*i* and is also available for download from the OTN site: http://technet.oracle.com/tech/xml

It is located in $ORACLE_HOME/xdk/cpp/classgen.

Using XML C++ Class Generator

The XML C++ Class Generator creates source files from an XML DTD. The Class Generator takes the Document Type Definition (DTD) and generates classes for each defined element. Those classes are then used in a C++ program to construct XML documents conforming to the DTD.

This is useful when an application wants to send an XML message to another application based on an agreed-upon DTD or as the back end of a web form to construct an XML document. Using these classes, C++ applications can construct, validate, and print XML documents that comply with the input DTD.

The Class Generator works in conjunction with the Oracle XML Parser for C++, which parses the DTD and passes the parsed document to the class generator.

> **See Also:** Chapter 3, "Oracle XML Components and General FAQs", "Using Oracle XML Components to Generate XML Documents: C++" on page 3-21.

External DTD Parsing

The XML C++ Class Generator can also parse an external DTD directly without requiring a complete (dummy) document. By using the Oracle XML Parser for C++ routine, `xmlparsedtd()`.

The provided command-line program `xmlcg` has a new '-d' option that is used to parse external DTDs. See "xmlcg Usage" on page 23-4.

Error Message Files

Error message files are provided in the mesg/ subdirectory. The messages files also exist in the `$ORACLE_HOME/oracore/mesg` directory. You may set the environment variable `ORA_XML_MESG` to point to the absolute path of the `mesg/` subdirectory although this not required.

XML C++ Class Generator Usage

Figure 23–1 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 Appendix E, "XDK for C++: Specifications and Cheat Sheet".

2. Two source files are output, a xxxxx.h header file and a xxxxx.cpp C++ file. These are named after the DTD file.

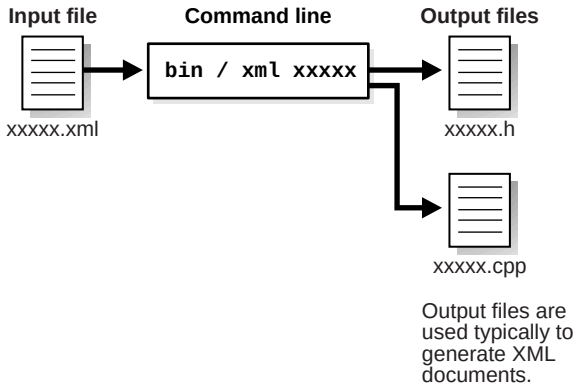
3. The output files are typically used to generate XML documents.

Constructors are provided for each class (element) that allow an object to be created in the following two ways:

- Initially empty, then adding the children or data after the initial creation
- Created with the initial full set of children or initial data

A method is provided for #PCDATA (and Mixed) elements to set the data and, when appropriate, set an element's attributes.

Figure 23–1 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 23–1 lists the xmlcg optional flags.

Table 23–1 xmlcg Optional Flags

xmlcg Optional Flags	Description
-d name	DTD - Input is an external DTD with the given name
-o directory	Output - 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 23–2 lists the files in the sample XML C++ Class Generator sample/ directory.

Table 23–2 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 IX

XDK for PL/SQL

Part IX describes how to access and use the XML Development Kit (XDK) for PL/SQL PL/SQL.

It contains the following chapters:

- Chapter 24, "Using XML Parser for PL/SQL"

FAQs are located at the end of this chapter.

24

Using 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 Oracle8*i* and is also available for download from the OTN site: http://technet.oracle.com/tech/xml.

It is located at $ORACLE_HOME/xdk/plsql/parser

What's Needed to Run XML Parser for PL/SQL

Appendix F, "XDK for PL/SQL: Specifications and Cheat Sheets" lists the specifications and requirements for running the XML Parser for PL/SQL. It also includes syntax cheat sheets.

Using XML Parser for PL/SQL (DOM Interface)

The XML Parser for PL/SQL makes developing XML applications with Oracle*8i* a simplified and standardized process. With the PL/SQL interface, Oracle shops familiar with PL/SQL can extend existing applications to take advantage of XML as needed.

Since the XML Parser for PL/SQL is implemented in PL/SQL and Java, it can run "out of the box" on the Oracle*8i* 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 API's developers need for processing XML documents:

- W3C-recommended Document Object Model (DOM)
- XSL-T and XPath recommendations

This makes writing custom applications that process XML documents straightforward in the Oracle*8i* environment, and means that a standards-compliant XML parser is part of the Oracle*8i* platform on every operating system where Oracle8i is ported.

Figure 24–1 shows the XML Parser for PL/SQL usage and parsing process diagram.

1. Make a newParser declaration to begin the parsing process for the XML document and DTD, if applicable.

   Table 24–1 lists available properties for the newParser procedure:

Table 24–1 XML Parser for PL/SQL: newParser() Properties

Property	Description
setValidationMode	Default = Not
setPreserveWhiteSpace	Default = Not
setDocType	Use if input type is a DTD
setBaseURL	Refers to other locations to the base locations, if reading from an outside source
showWarnings	Turns warnings on or off.

2. The XML and DTD can be input as a file, varchar buffer, or CLOB. The XML input is called by the following procedures:
   - parse() if the XML input is a file
   - parseBuffer() if the XML input is an varchar buffer
   - `parserClob()` if the XML input is a CLOB

   If a DTD is also input, it is called by the following procedures:
   - `parseDTD()` if the input is an DTD file
   - `parseDTDBuffer()` if the DTD input is an varchar buffer
   - `parserDTDClob()` if the DTD input is a CLOB

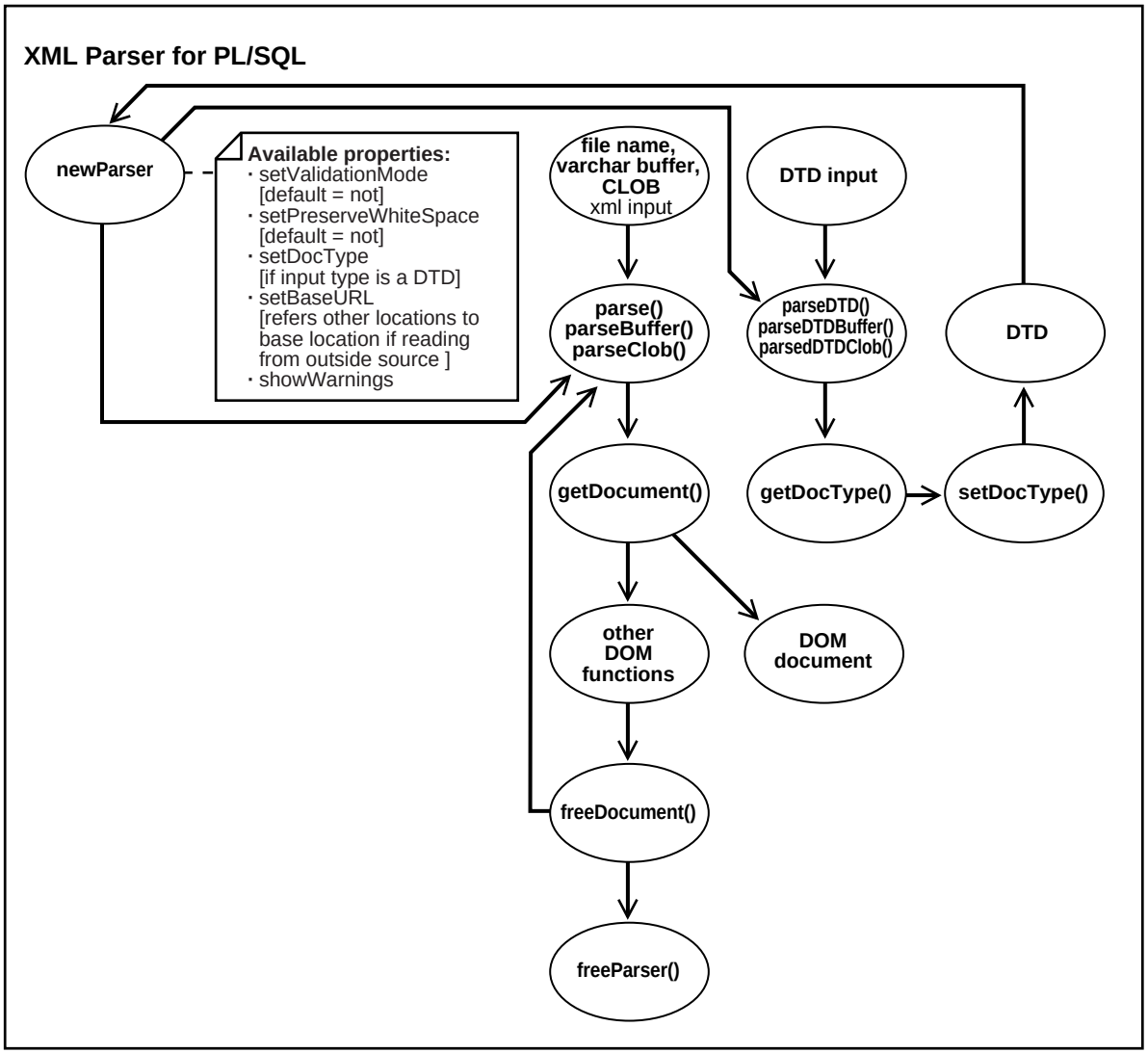
For the XML Input: For an XML input, the parsed result from `Parse()`, `ParserBuffer()`, or `ParserClob()` procedures is sent to `GetDocument()`.

3. `getDocument()` procedure performs the following:
   - Outputs the parsed XML document as a DOM document typically to be used in a PL/SQL application, or
   - Applies other DOM functions, if applicable. See *Oracle8i XML Reference* for a list of available optional DOM functions.
4. Use `freeDocument()` function to free up the parser and parse the next XML input
5. Use `freeParser()` to free up any temporary document structures created during the parsing process

For the DTD input: The parsed result from `parseDTD()`, `parseDTDBuffer()`, or `parseDTDClob()` is used by `getDocType()` function.

6. `getDocType()` then uses `setDocType()` to generate a DTD object.
7. The DTD object can be fed into the parser using `setDocType()` to override the associated DTD.

Figure 24–1 XML Parser for PL/SQL Functionality (DOM Interface)



XML Parser for PL/SQL: Default Behavior

The following is the default behavior for XML Parser for PLSQL XML:

- A parse tree which can be accessed by DOM APIs is built
- The parser is validating if a DTD is found, otherwise it is non-validating
- Errors are not recorded unless an error log is specified; however, an application error will be raised if parsing fails

The types and methods described in this manual are supplied with the PLSQL package xmlparser().

Using the XML Parser for PL/SQL: XSL-T Processor (DOM Interface)

Extensible Stylesheet Language Transformation, abbreviated XSLT (or XSL-T), describes rules for transforming a source tree into a result tree. A transformation expressed in XSLT is called a stylesheet.

The transformation specified is achieved by associating patterns with templates defined in the stylesheet. A template is instantiated to create part of the result tree.

This PLSQL implementation of the XSL processor follows the W3C XSLT working draft (rev WD-xslt-19990813) and includes the required behavior of an XSL processor in terms of how it must read XSLT stylesheets and the transformations it must effect.

The types and methods described in this document are made available by the PLSQL package, `xslprocessor()`.

Figure 24–2 shows the XML Parser for PL/SQL XSL-T Processor main functionality.

1. The Process Stylesheet process receives input from the XML document and the selected Stylesheet which m,ay or may nto be indicated in the XML document. Bot the stylesheet and the XML document can be any of the following types:

   - File name
   - Varchar buffer
   - CLOB

   The XML document can be input 1 through n times.

2. The parsed XML document inputs `XSLProcessor.processXSL(xslstylesheet,xml instance)` procedure, where:

   - XML document is indicated in the "xml instance" argument
   - Stylesheet input is indicated in the "xslstylesheet" argument

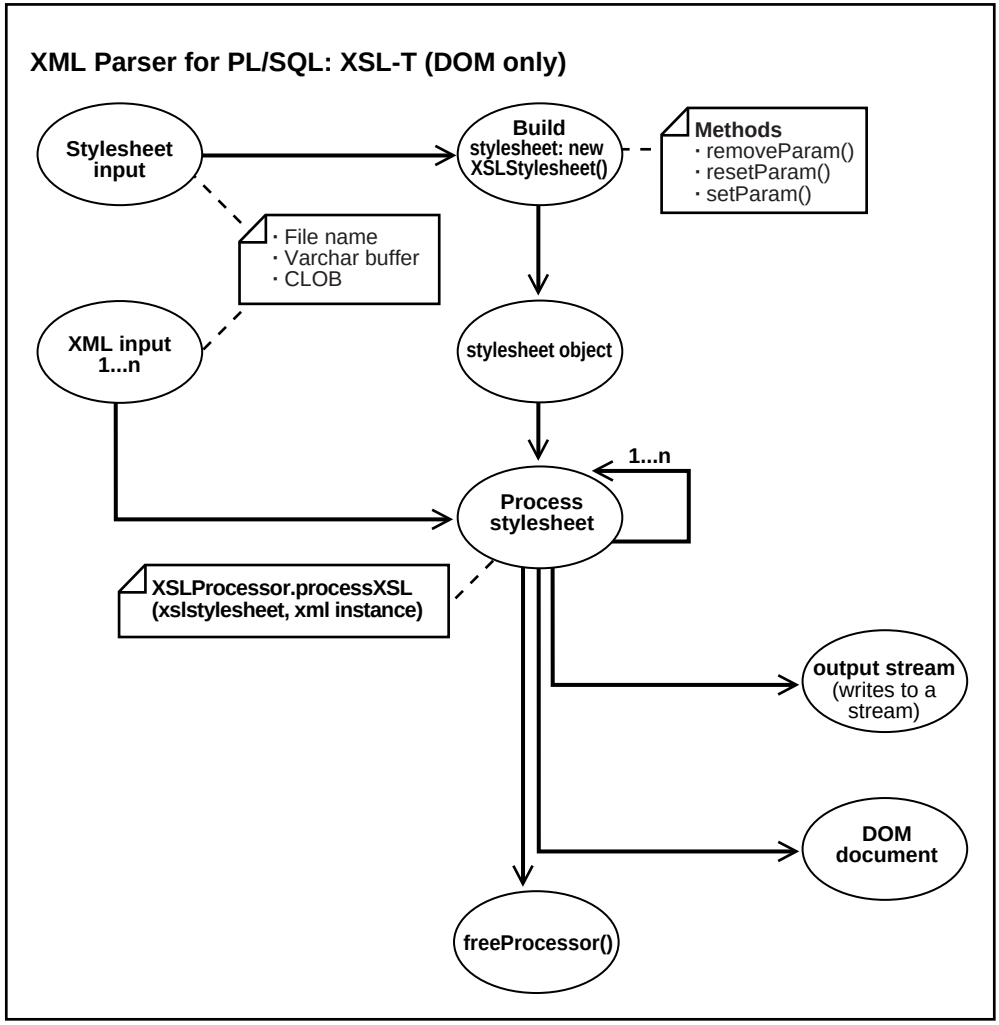
3. Build the stylesheet using the Stylesheet input to the `XSLStylesheet()` procedure. The following methods are available for this procedure:

   - `removeParam()`
   - `resetParam()`
   - `setParam()`

This produces a stylesheet object which then inputs the "Process Stylesheet" step using procedure, `XSLProcessor.processXSL(xslstylesheet,xml instance)`.

4. The "Process stylesheet" process can be repeated 1 through n times. In other words, the same stylesheet can be applied to multiple parsed XML documents to transform them wither into an XML document, HTML document, or other text based format.
5. The resulting parsed and transformed document is output either as a stream or a DOM document.
6. When the XSL-T process if complete, call the freeProcessor() procedure to free up any temporary structures and the XSLProcessor procedures used in the XSL transformation process.

Figure 24–2 "XML Parser for PL/SQL: XSL-T processor (DOM Interface)



XML Parser for PL/SQL: XSLT Processor — Default Behavior

The following is the default behavior for the XML Parser for PL/SQL XSLT Processor:

- A result tree which can be accessed by DOM APIs is built

- Errors are not recorded unless an error log is specified; however, an application error will be raised if parsing fails

Using XML Parser for PL/SQL Examples in sample/

Setting Up the Environment to Run the sample/ Sample Programs

The $ORACLE_HOME/xdk/plsql/parser/sample/ directory contains two sample XML applications:

- domsample
- 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 admin privileges, such as, 'internal':

   For example

   ```
   % sqlplus
   SQL> connect internal
   ```

3. A password might be required for 'internal' or the appropriate user with admin privileges. Contact your System Administrator, DBA, or Oracle support, if you cannot login with admin 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 nam
'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://technet.oracle.com

XML Parser for PL/SQL Example 1: XML — family.xml

This XML file inputs `domsample.sql`.

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

XML Parser for PL/SQL Example 2: DTD — family.dtd

This DTD file is referenced by XML file, `family.xml`.

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

XML Parser for PL/SQL Example 3: XSL — iden.xsl

This XSL file inputs the xslsample.sql.

```
<?xml version="1.0"?>

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

```
  </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

XML Parser for PL/SQL Example 4: PL/SQL — domsample.sql

```
-- This file demonstrates a simple use of the parser and DOM API.
-- The XML file that is given to the application is parsed and the
-- elements and attributes in the document are printed.
-- It shows you how to set the parser options.

set serveroutput on;
create or replace procedure domsample(dir varchar2, 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 5: PL/SQL — xslsample.sql

```
-- This file demonstates a simple use of XSL-T transformation capabilities.
-- The XML and XSL files that are given to the application are parsed,
-- the transformation specified is applied and the transformed document is
-- written to a specified result file.
-- It shows you how to set the parser options.

set serveroutput on;
create or replace procedure xslsample(dir varchar2, xmlfile varchar2,
                                      xslfile varchar2, resfile varchar2,
                                      errfile varchar2) is
p xmlparser.Parser;
xmldoc xmldom.DOMDocument;
xmldocnode xmldom.DOMNode;
proc xslprocessor.Processor;
ss xslprocessor.Stylesheet;
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 (FAQs): XML Parser for PL/SQL

Exception in Thread Parser Error

Question

When I try to use the `oraxsl` I get the following: Exeception in thread "main":

```
java.lang.NoClassDefFoundError" oracle/xml/parser/v2/oraxsl.
```

How do I fix this?

Answer

Can you provide more details as to your configuration and usage? If you are running outside the database you need to make sure the xmlparserv2.jar is explicitly in your CLASS_PATH not simply its directory. If from the database you need to make sure it has been properly loaded and that JServer initialized.

Encoding '8859_1' is not currently supported by the JavaVM

Question

I parsed my XML document using the XML Parser for PL/SQL and modified some of the node values of the DOMDocument by using "setNodeValue". When I tried to write the modified DOMDocument to buffer or file using "writeToBuffer" or "writeToFile, both commands gave me the following error:

```
ORA-20101: Error occurred while accessing a file or URL: Encoding '8859_1' is
not currentlysupported by the JavaVM
```

Comment

I just reinstalled initjvm.sql and also installed the latest version of the XML Parser for PL/SQL. Everything is working fine.

xmldom.GetNodeValue in PL/SQL

Question

I cannot get the element value using the PL/SQL XMLDOM. Here is the code fragment:

```
...nl := xmldom.getElementsByTagName(doc, '*');
len := xmldom.getLength(nl)
;-- loop through elements
   for i in 0..len-1 loop      n := xmldom.item(nl, i);
     elename := xmldom.getNodeName(n);
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, for example:

```
...t xmldom.DOMText;
...t := xmldom.makeText(n);
eleval := xmldom.getNodeValue(t);
```

What am I doing wrong?

Comment

I found the answer to my own question. To get the text node value associated with the element node, you must perform additional node navigation via xmldom.getFirstChild(n).

To illustrate, change printElements() in DOMSample.sql as follows:

```
begin
-- get all elements
nl := xmldom.getElementsByTagName(doc, '*');
 len := xmldom.getLength(nl);
  -- loop through elements
for i in 0..len-1 loop      n := xmldom.item(nl, i);
      dbms_output.put(xmldom.getNodeName(n));
      -- get the text node associated with the element node
      n := xmldom.getFirstChild(n);
     if xmldom.getNodeType(n) = xmldom.TEXT_NODE then         dbms_
output.put('=' &#0124; &#0124; xmldom.getNodeValue(n));
      end if;
     dbms_output.put(' ');
   end loop;
   dbms_output.put_line('');
end printElements;
```

This produces the following output:

The elements are:

```
family member=Sarah member=Bob member=Joanne member=Jim
```

The attributes of each element are:

```
family:familylastname val=Smithmember:membermemberid val=m1member:membermemberid
val=m2member:membermemberid val=m3 mom val=m1 dad val=m2member:membermemberid
val=m4 mom val=m1 dad val=m2
```

XDK for PL/SQL Toolkit

Question

I downloaded XDK for PL/SQL but it requires OAS. Do you have any idea how to run this in an IIS environment?

Answer

If you're going to use IIS, it would be better to use the XML Parser for Java V2.You'll need Oracle8*i*.

Parsing DTD contained in a CLOB (PL/SQL) XML

Question

I am having problems parsing a DTD file contained in a CLOB. I used the API, "xmlparser.parseDTDClob", provided by the XML Parser for PL/SQL.

The following error was thrown:

```
"ORA-29531: no method parseDTD in class oracle/xml/parser/plsql/XMLParserCover".
```

The procedure `xmlparser.parseDTDClob` calls a Java Stored Procedure `xmlparsercover.parseDTDClob`, which in turn calls another Java Stored Procedure `xmlparsercover.parseDTD`.

I have confirmed that the class file, "`oracle.xml.parser.plsql.XMLParserCover`", has been loaded into the database, and that it has been published. So the error message does not make sense. The procedure used to call "xmlparser.parseDTDClob" is:

```
create or replace procedure parse_my_dtd as p xmlparser.parser; l_clob clob;
begin   p := xmlparser.newParser;   select content into l_clob from dca_
documents where doc_id = 1;   xmlparser.parseDTDClob(p,l_clob,'site_template');
end; API Documentation for xmlparser.parseDTDClob:

parseDTDClob PURPOSE   Parses the DTD stored in the given clob SYNTAX
```

```
PROCEDURE parseDTDClob(p Parser, dtd CLOB, root VARCHAR2); PARAMETERS   p
(IN)- parser instance dtd      (IN)- dtd clob to parse root     (IN)- name
of the root element RETURNS   Nothing COMMENTS
```

Any changes to the default parser behavior should be effected before calling this procedure. An application error is raised if parsing failed, for some reason. Description of the table dca_documents:

```
DOC_ID          NOT NULL   NUMBER  DOC_NAME       NOT NULL   VARCHAR2(350)  DOC_
TYPE                    VARCHAR2(30)
 DESCRIPTION                VARCHAR2(4000)  MIME_TYPE
VARCHAR2(48)  CONTENT         NOT NULL   CLOB  CREATED_BY     NOT NULL
VARCHAR2(30)  CREATED_ON      NOT NULL   DATE  UPDATED_BY     NOT NULL
VARCHAR2(30)  UPDATED_ON      NOT NULL   DATE
```

The contents of the dtd:

```
<!ELEMENT site_template (component*)> <!ATTLIST site_template template_id CDATA
#REQUIRED> <!ATTLIST site_template template_name CDATA #REQUIRED> <!ELEMENT
component (#PCDATA)> <!ATTLIST component component_id ID #REQUIRED> <!ATTLIST
component parent_id ID #REQUIRED> <!ATTLIST component component_name ID
#REQUIRED>
```

Answer

This is a known issue in the 1.0.1 release of the XML Parser for PL/SQL. Here is the workaround.

1. Make a backup of ./plsqlxmlparser_1.0.1/lib/sql/xmlparsercover.sql
2. In line 18 in xmlparsercover.sql, change the string:
   oracle.xml.parser.plsql.XMLParserCover.parseDTD to
   oracle.xml.parser.plsql.XMLParserCover.parseDTDClob
3. Verify that Line 18 now reads: procedure parseDTDClob(id varchar2, dtd CLOB, root varchar2, err in out varchar2) is language java name 'oracle.xml.parser.plsql.XMLParserCover.parseDTDClob(java.lang.String, oracle.sql.CLOB, java.lang.String, java.lang.String[])';
4. Save the file
5. Rerun xmlparsercover.sql in SQL*Plus Assuming you've loaded XMLParserV2 release 2.0.2.6 into the database, this should solve your problem.

XML Parser for PL/SQL

Question

I have just started using XML Parser for PL/SQL. I am have trouble getting the text between the begin tag and the end tag into a local variable. Do you have examples?

Answer

You just have to use the following:

```
selectSingleNode("pattern");
getNodeValue()
```

Remember, if you are trying to get value from a Element node, you have to move down to the #text child node, for example, getFirstChild.getNodeValue()

Suppose you need to get the text contained between the starting and ending tags of a xmldom.DOMNode n. The following 2 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

Security: ORA-29532, Granting JavaSysPriv to User

Question

We are using the XML Parser for PLSQL and are trying to parse an XML document. We are getting a Java security error:

```
ORA-29532: Java call terminated by uncaught Java exception:
java.lang.SecurityException ORA-06512: at "NSEC.XMLPARSERCOVER", line 0
ORA-06512: at "NSEC.XMLPARSER", line 79 ORA-06512: at "NSEC.TEST1_XML line 36
ORA-06512: at line 5
```

Do we need to grant to user? The syntax appears correct. We also get the error when we run the demo.

Answer

If the document you are parsing contains a <!DOCTYPE which has a System URI with a protocol like file:/ / / or http:/ / / then you to grant an appropriate privilege to your current database user to be able to "reach out of the database", so to speak, and open a stream on the file or URL.CONNECT SYSTEM/MANAGER

```
GRANT JAVAUSERPRIV, JAVASYSPRIV TO  youruser;
```

should do it.

Installing XML Parser for PL/SQL: JServer(JVM) Option

Question

I have downloaded and installed the plxmlparser_V1_0_1.tar.gz. The readme said to use loadjava to upload xmlparserv2.jar and plsql.jar in order. I tried to load xmlparserv2.jar using the following command:

loadjava -user test/test -r -v xmlparserv2.jar

to upload the jar file into Oracle8i. After much of the uploading, I got the following error messages:

```
identical: oracle/xml/parser/v2/XMLConstants is unchanged from previously loaded
fileidentical: org/xml/sax/Locator is unchanged from previously loaded
fileloading  : META-INF/MANIFEST.MFcreating : META-INF/MANIFEST.MFError while
creating resource META-INF/MANIFEST.MF    ORA-29547: Java system class not
available: oracle/aurora/rdbms/Compilerloading  :
oracle/xml/parser/v2/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.

XML Parser for PL/SQL: domsample

Question

I am trying to execute domsample on dom1151. This is an example that is provided with installation. XML file family.xml is present in the directory /hrwork/log/pqpd115CM/out.

Still I am getting the following error.

Usage of domsample is domsample(dir, inpfile, errfile)

```
SQL>
begin
domsample('/hrwork/log/pqpd115CM/out','family.xml','errors.txt');
  end;
  /
Error generated :
begin
*
ERROR at line 1:
ORA-20100: Error occurred while parsing: No such file or directory
ORA-06512: at "APPS.XMLPARSER", line 22
ORA-06512: at "APPS.XMLPARSER", line 69
ORA-06512: at "APPS.DOMSAMPLE", line 80
ORA-06512: at line 2
```

Answer

From your description it sounds like you have not completed all of the steps in the sample/Readme without errors. After confirming the xmlparserv2.jar is loaded, carefully complete the steps again.

XML in CLOBs

Question

In 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 Oracle8*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 like:

- xmlparser.parseCLOB()

As well as methods like:

- xslProcessor.selectNodes() to find what part of the doc you are looking for
- xmldom.* methods to manipulate the content of the XML Doc
- xmldom.writeToCLOB() to write it back

If you wanted to do surgical updates on the text of the CLOB, you would have to use DBMS_LOB.* routines, but this would be tricky unless the changes being made to the content don't involve any growth or shrinkage of the number of characters.

Out of memory errors in oracle.xml.parser

Question

Out of memory errors in oracle.xml.parser

```
last entry at 2000-04-26 10:59:27.042:
```

```
VisiBroker for Java runtime caught exception:
java.lang.OutOfMemoryError
   at oracle.xml.parser.v2.XMLAttrList.put(XMLAttrList.java:251)
   at oracle.xml.parser.v2.XMLElement.setAttribute(XMLElement.java:260)
   at oracle.xml.parser.v2.XMLElement.setAttribute(XMLElement.java:228)
   at cars.XMLServer.processEXL(XMLServer.java:122)
```

It's trying to create a new XML attribute and crashes with OutOfMemoryError.

We are parsing a 50Mb XML file. We have upped the java_pool_size to 150Mb with a shared_pool_size of 200Mb.

Answer

You should not be using the DOMParser for parsing a 50Mb XML file. You need to look at the SAXParser which parses files of arbitrary size because it does not create an in-memory tree of nodes as it goes.

Which parser are you using, SAX or DOM - if you are using DOM, you should seriously consider moving to SAX which processes the XML file sequentially instead of trying to build an in-memory tree that represents the file.

Using SAX we process XML files in excess of 180Mb without any problems and with very low memory requirements.

Rule of thumb for DOM and SAX:

DOM:

- DOM is very good when you need some sort of random access
- DOM consumes more memory
- DOM is also good when you are trying to transformations of some sort
- DOM is also good when you want to have tree iteration and want to walk through the entire document tree
- See if you can use more attributes over elements in your XML (to reduce the pipe size)

SAX:

- SAX is good when data comes in a streaming manner (using some input stream)

Is There a PL/SQL Parser Based on C?

Question

Is there a PL/SQL parser that is based on C?

Answer

There is not one currently but there are plans to provide the PL/SQL parser on top of the C version.

Memory Requirements When Using the Parser for PL/SQL

Question

What are the memory requirements for using the PL/SQL Parser?

Answer

While the memory use is directly dependent on the document size, it should also be realized that the PL/SQL parser uses the Java parser and thus the Oracle JServer is being run. JServer typically requires 40-60MB depending on its configuration.

JServer (JVM) , Is It Needed to Run XML Parser for PL/SQL?

Question

Do I need to install JServer to run the XML Parser for PL/SQL?

Answer

Yes, if you are running the parser in the database, you do need JServer because the PL/SQL Parser currently uses the XML Parser for Java under the covers. JServer exists in both the Standard and Enterprise versions. A forthcoming version of XML Parser for PL/SQL using C underneath is being developed for applications that do not have access to a Java Virtual Machine (JVM).

Using the DOM API

Question - What does the XML Parser for PL/SQL do?

Answer

The XML parser accepts any XML document giving you a tree-based API (DOM) to access or modify the document's elements and attributes. It also supports XSLT which allows transformation from one XML document to another.

Question - Is it possible to dynamically set the encoding in the XML document?

Answer

No, you need to include the proper encoding declaration in your document as per the specification. You cannot use setCharset(DOMDocument) to set the encoding for the input of your document. SetCharset(DOMDocument) is used with oracle.xml.parser.v2.XMLDocument to set the correct encoding for the printing.

Question - How do I get the number of elements in a particular tag using the parser?

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.

Question - How do I parse a string?

A: We do not currently have any method that can directly parse an XML document contained within a String. You can use

- function parse (Parser, VARCHAR2) to parse XML data stored in the given URL or the given file,
- function parseBuffer (Parser, VARCHAR2) to parser XML data stored in the given buffer, or
- function parseCLOB (Parser, VARCHAR2) to parse XML data stored in the give CLOB.

Question - How do I display my XML document?

Answer

If you are using IE5 as your browser you can display the XML document directly. Otherwise, you can use our XSLT processor in v2 of the parser to create the HTML document using an XSL Stylesheet. Our Java Transviewer bean also allows you to view your XML document.

Question - How do I write the XML data back using a special character sets?

Answer

You can specified the character sets for writing to a file or a buffer. Writing to a CLOB will be using the default character set for the database that you are writing to. Here are the methods to use:

- procedure writeToFile(doc DOMDocument, fileName VARCHAR2, charset VARCHAR2);
- procedure writeToBuffer(doc DOMDocument, buffer IN OUT VARCHAR2, charset VARCHAR2);
- procedure writeToClob(doc DOMDocument, cl IN OUT CLOB, charset VARCHAR2);

Question - How do I to get ampersand from characterData?

Answer

You cannot have "raw" ampersands in XML data. You need to use the entity, & instead. This is defined in the XML standard.

Question - How do I generate a document object from a file?

Answer

Check out the following example:

```
inpPath VARCHAR2;
inpFile VARCHAR2;
p xmlparser.parser;
doc xmldom.DOMDocument;
```

```
-- initialize a new  parser object;
p := xmlparser.newParser;
-- parse the file
xmlparser.parse(p, inpPath || inpFile);
-- generate a document object
doc := xmlparser.getDocument(p);
```

Question - Can the parser run on Linux?

Answer

As long as a 1.1.x or 1.2.x JavaVM for Linux exists in your installation, you can run the Oracle XML Parser for Java there. Otherwise, you can use the C or C++ XML Parser for Linux.

Question - How do I perform a >,<,>=, or <= comparison using the XML Parser v2?

Answer

You need to use the entities < for < and > for >.

Question -Is support for Namespaces and Schema included?

Answer

The current XML parsers support Namespaces. Schema support will be included in a future release.

Question -My parser doesn't find the DTD file.

Answer

The DTD file defined in the <!DOCTYPE> declaration must be relative to the location of the input XML document. Otherwise, you'll need to use the setBaseDir(Parser, VARCHAR2) functions to set the base URL to resolve the relative address of the DTD.

Question - Can I validate an XML file using an external DTD?

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.

Question - Do you have DTD caching?

Answer

Yes, DTD caching is optional and it is not enabled automatically.

Question - How do I get the DOCTYPE tag into the XMLDocument after its parsed?

Answer

You need to do some preprocessing to the file, and then put it through the DOMParser again, which will produce a valid, well-formed XMLDocument with the DOCTYPE tag contained within.

Question - How does the XML DOM parser work?

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.

Question - How do I create a node whose value I can set later?

Answer

If you check the DOM spec referring to the table discussing the node type, you will find that if you are creating an element node, its nodeValue is to be null and hence cannot be set. However, you can create a text node and append it to the element node. You can store the value in the text node.

Question - 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.

Question - 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.

Question - I am using XML parser with DOM but I cannot get the actual data. What is wrong?

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.

Question - Can the XML Parser for PL/SQL handle stylesheets that produce non-XML documents such as HTML?

Answer

Yes it can.

Using the Sample

Question - 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.

XML Parser for PL/SQL: Parsing DTD in a CLOB

Question

I am having problems parsing a DTD file contained in a CLOB. I used the API, "xmlparser.parseDTDClob", provided by the XML Parser for PL/SQL.

The following error was thrown:

```
"ORA-29531: no method parseDTD in class oracle/xml/parser/plsql/XMLParserCover"
```

I managed to work out the following:

The procedure xmlparser.parseDTDClob calls a Java Stored Procedure xmlparsercover.parseDTDClob, which in turn calls another Java Stored Procedure xmlparsercover.parseDTD.

I have confirmed that the class file -"oracle.xml.parser.plsql.XMLParserCover" has been loaded into the database, and that it has been published. So the error message does not make sense.

I am not able to figure out whether I am doing it right or whether this is a bug in the parser API.

```
The procedure use to call "xmlparser.parseDTDClob" :
-----------------------------------------------------
create or replace procedure parse_my_dtd as
p xmlparser.parser;
l_clob clob;
begin
  p := xmlparser.newParser;
  select content into l_clob from dca_documents where doc_id = 1;
  xmlparser.parseDTDClob(p,l_clob,'site_template');
```

```
end;
```

API Documentation for xmlparser.parseDTDClob:

```
parseDTDClob
PURPOSE
  Parses the DTD stored in the given clob
SYNTAX
  PROCEDURE parseDTDClob(p Parser, dtd CLOB, root VARCHAR2);
PARAMETERS
  p        (IN)-  parser instance
 dtd      (IN)-  dtd clob to parse
 root     (IN)-  name of the root element
RETURNS
  Nothing
COMMENTS
```

Any changes to the default parser behavior should be effected before calling this procedure. An application error is raised if parsing failed, for some reason.

Description of the table dca_documents:

```
 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 (a)

It appears to be a typo in the "xmlparsercover.sql" script which is defining the Java Stored Procedures that wrap the XMLParser. It mentions the Java method name "parseDTD" in the 'is language java name' part when "parseDTD" should be "parseDTDClob" (case-sensitive).

If you:

1. Make a backup copy of this script
2. Edit the line that reads:

```
procedure parseDTDClob(id varchar2,
dtd CLOB, root varchar2, err in out varchar2) is language java name
'oracle.xml.parser.plsql.XMLParserCover.parseDTD (java.lang.String,
oracle.sql.CLOB, java.lang.String, java.lang.String[])';
```

   to say:

```
procedure parseDTDClob(id varchar2,
dtd CLOB, root varchar2, err in out varchar2) is language java name
'oracle.xml.parser.plsql.XMLParserCover.parseDTDClob
(java.lang.String, oracle.sql.CLOB, java.lang.String,
java.lang.String[])';
```

   that is, change the string:

```
'oracle.xml.parser.plsql.XMLParserCover.parseDTD
```

   to

```
'oracle.xml.parser.plsql.XMLParserCover.parseDTDClob
```

   and rerun the xmlparsercover.sql script you should be in business.

I filed a bug 1147031 to get this typo corrected in a future release.

Note: Your DTD had syntactic errors in it, but I was able to run the following without problem after making the change above:

```
declare
   c clob;
   v varchar2(400) :=
'<!ELEMENT site_template  (component* )>
<!ATTLIST site_template  template_name CDATA  #IMPLIED
                         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 (b)

What do u want to do with the LOB. The LOB can either be a temporary LOB or a persistant lob. In case of persistant lobs, u need to insert the value into a table. In case of temp lob u can instantiate it in ur 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 *Oracle8i Application Developer's Guide - Large Objects (LOBs)*. There are 6 books one for each language access (C(OCI), Java,PL / SQL, Visual Basic, Pro*C / C++, Pro*Cobol)) and it is quite comprehensive. If this is PL / SQL, I believe you can just do the following:

```
myClob CLOB = clob();
```

I have tried the DBMS_LOB.createtemporary() which works.

Answer (c)

Here's what you need to do if you are using LOBs with AQ:

1. Create an ADT with one of the fields of type CLOB.

   ```
   create type myAdt (id NUMBER, cdata CLOB);
   ```

   The queue table must be declared to be of type myAdt

2. Instantiate the object - use empty_clob() to fill the LOB field

   ```
   myMessage := myAdt(10, EMPTY_CLOB();
   ```

3. Enqueue the message

   ```
   clob_loc clob;
   enq_msgid RAW(16);
   DBMS_AQ.enqueue('queue1', enq_opt, msg_prop, myMessage, enq_msgid)
   ```

4. Get the LOB locator

   ```
   select t.user_data.cdata into clob_loc
   from qtable t  where t.msgid
   = enq_msgid;
   ```

5. Populate the CLOB using dbms_lob.write
6. Commit

There is an example of this is in the *Oracle8i Application Developer's Guide - Advanced Queuing*. If you are using the Java API for AQ, the procedure is slightly more complicated.

Errors When Parsing a Document

I downloaded the javaparser v2 and the xml parser utility and I'm using the PLSQL parser interface. I have an XML file that is a composite of three tags and when parsing it generates the following error:

```
ORA-20100: Error occurred while parsing: Unterminated string
```

When I separate the document into individual tags 2 are ok the third generates this error:

```
ORA-20100: Error occurred while parsing: Invalid UTF8 encoding
```

1. Why is the error different when separating the data?
2. I have not been able to find an "unterminated string" in the document.

3. I'm fairly anxious since this is the only way the data is coming and I don't have time to figure out another parser.

Answer

If you document is the "composite of three tags" then it is not a well-formed document as it has more than one root element. Try putting a start and end tag around the three.

PLXML: Parsing a Given URL?

Question

I'am working with the XML parser for PL/SQL on NT.According to your Parser API documentation it is possibleto 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 urlraises ORA-29532: ... java.io.FileNotFoundException

Can you help me ? Can you give an example of a call ?Are there prerequisites i have to think of ?

Answer

To access external URLs, you need set up your proxy host and port. For example using this type of syntax:

```
java -Dhttp.proxyHost=myproxy.mydomain.com -Dhttp.proxyPort=3182DOMSample
myxml.xml
```

Using XML Parser to Parse HTML?

Question

We need to parse HTML files as follows:

1. Find each "a href"
2. For each a href found, extract the file/pathname being linked to
3. Substitute a database procedure call for the a href, passing the file/pathname as a parameter.

Does it make sense to use the PL/SQL XML parser to do this? If so, how easy/hard would it be, and how can we find out how to do this?

Answer

Since HTML files aren't necessary well formed XML documents, are you sure you want to use XML parser? Won't PERL be a better choice?I'm not sure whether PL/SQL parser supports the following methods but just for your information:

1. getElementsByTagName() retrieves all matching nodes.
2. getNodeValue() will return a string.
3. setNodeValue() sets node values.

Answer b

It supports those methods, but not over an ill-formed HTML file.

Oracle 7.3.4: Moving Data to a Web Browser (PL/SQL)

Question

I'm trying to get the data to a web browser in the client side while all the processing has to take place on the server (oracle 7.3.4), using:

- XML Parser for PL/SQL
- XSQL servlet

Are these two components sufficient to get the job done?

Answer

Dependencies for XSQL Page Processor states:

- Oracle XML Parser V2 R2.0.2.5
- Oracle XML-SQL Utility for Java
- Web server supporting Java Servlets
- JDBC driver

You'll also need XSQL Page Processor itself.

Oracle 7.3.4 and XML

Question

Does the XML Parser for Java,V2, work with Oracle 7.3.4.?

Is XML- SQL Utility part of XML Parser for Java,V2, or does it need to be downloaded separately.

Answer

1. The XML Parser for Java, V2 works with 7.3.4 as long as you have the proper JDBC driver and run it in a VM on a middle tier or client.
2. The XML-SQL Utility includes a copy of the v2 parser in its download, as it requires it.

getNodeValue(): Getting the Value of DomNode

Question

I am having problems obtaining the value between XML tags after using xmlparser(). Below is code from the DOMSAMPLE.SQL example:

```
-- loop through elementsfor i in 0..len-1 loop   n := xmlparser.item(nl, i);
   dbms_output.put(xmlparser.getNodeName(n)
```

Comment

I encountered the same problem. I found out that getNodeValue() on Element Node returns null. getNodeValue() on the *text* node returns the value.

Retrieving all Children or Grandchilden of a Node

Question

Is there a way to retrieve all childrenor 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 rootedas the element in "theElement".

A

An XML Primer

This Appendix contains the following sections:

What is XML?

XML, eXtensible Markup Language, is the standard way to identify and describe data on the web. It is widely implementable and easy to deploy.

XML is a human-readable, machine-understandable, general syntax for describing hierarchical data, applicable to a wide range of applications, databases, e-commerce, Java, web development, searching, and so on.

Custom tags enable the definition, transmission, validation, and interpretation of data between applications and between organizations.

W3C XML Recommendations

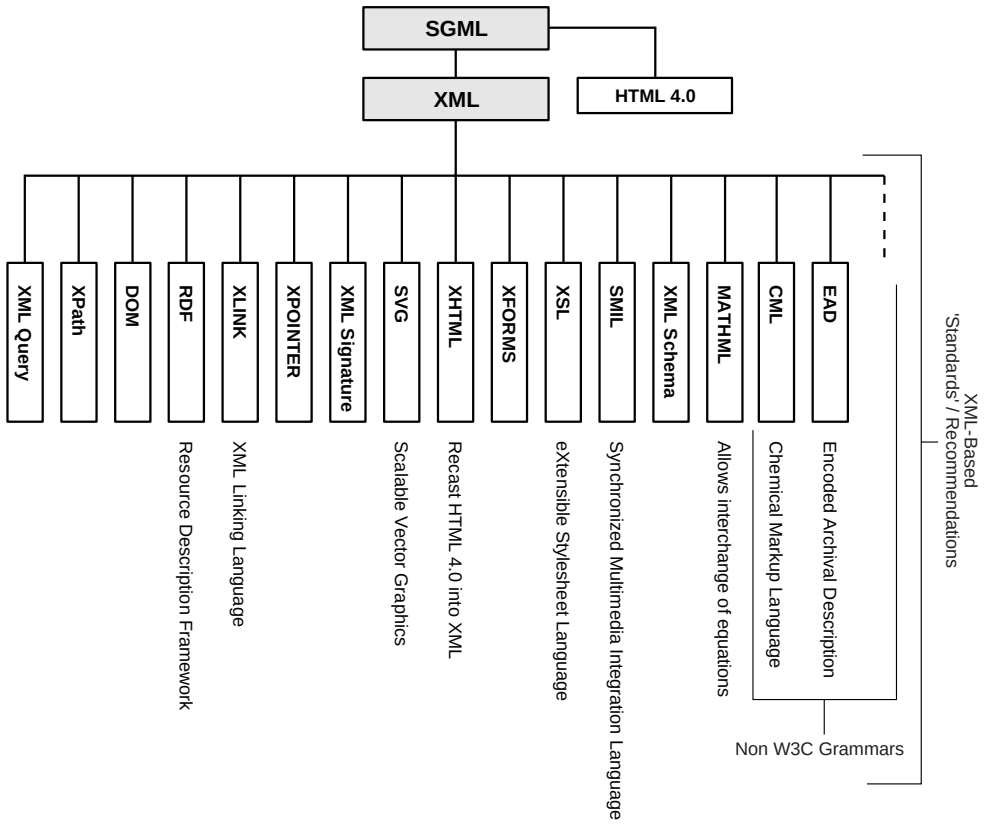
The World Wide Web Consortium (W3C) XML recommendations are an ever-growing set of interlocking specifications.

- *XML 1.0* was recommended by W3C in February 1998. It has resulted numerous additional W3C Working Groups, a Java Platform Extension Expert Group, and the XML conversion of numerous data interchange standards such as Electronic Data Interchange (EDI). The next version of HTML will be an XML application known as xHTML.
- *XML Namespaces.* Another W3C recommendation aimed at removing element ambiguity in multi-namespace well-formed XML applications.
- *XML Query*. The W3C standards effort to specify a query language for XML documents.
- *XML Schema*. The W3C standards effort to add simple and complex datatypes to XML documents and replace the functionality of DTDs with an XML Schema definiton XML document.
- *XSL*. XSL consists of two W3C recommendations:
  - XSL Transformations for transforming one XML document into another
  - XSL Formating Objects for specifying the presentation of an XML document
- *XPath*. XPath is the W3C recommendation that specificies the data model and grammar for navigating an XML document utilized by XSL-T, XLink, and XML Query.
- *XPointer.* XPointer is the W3C recomendation that specifies the identification of individual entities or fragments within an XML document using XPath navigation. This W3C proposed recommendation is defined at http://www.w3.org/TR/WD-xptr.

- *DOM*. The W3C recommendation that specifies the Document Object Model of an XML Document including APIs for programmatic access.

The XML family of applications is illustrated in Figure 24–3.

Figure 24–3 The XML Family of Applications ('Including XML-Based Standards')



XML Features

The following bullets describe XML features:

- **Data Exchange, From Structured to Unstructured Data**: XML enables a universal standard syntax for exchanging data. XML specifies a rigorous, text-based way to represent the structure inherent in data so that it can be authored and interpreted unambiguously. Its simple, tag-based approach leverages developers' familiarity of HTML but provides a flexible, extensible mechanism that can handle the gamut of "digital assets" from highly structured database records to unstructured documents and everything in between. " W3C
- **SGML Was Designed Specifically for Documents - XML is Designed for Potentially Any Data**: The SGML markup language was specifically designed for documents. Web-centric XML is like a toolkit that can be used to write other languages. It is not designed for documents only. Any data that can be described in a tree can be programed in XML.
- **A Class of Data Objects - A Restricted Form of SGML**: www.oasis-open.org describes XML as follows: "... XML, describes a class of data objects called XML *documents* and partially describes the behavior of computer programs which process them. XML is an application profile or restricted form of SGML, the Standard Generalized Markup Language. By construction, XML documents are conforming SGML documents."
- **XML's Many Uses...**: A W3C.org press release describes XML as follows: "... XML is primarily intended to meet the requirements of large-scale Web content providers for industry-specific markup, vendor-neutral data exchange, media-independent publishing, one-on-one marketing, workflow management in collaborative authoring environments, and the processing of Web documents by intelligent clients.
- **Metadata**. XML is also finding use in certain metadata applications.
- **Internationalization**. "XML is fully internationalized for both European and Asian languages, with all conforming processors required to support the Unicode character set in both its UTF-8 and UTF-16 encodings..." Its primary use is for electronic publishing and data interchange ..."
- **Parsed or Unparsed Storage Entities**: From the W3C.org XML specification proposal: "... XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form the character data in the document, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure.

- **XML Processor Reads XML Documents**. "... XML provides a mechanism to impose constraints on the storage layout and logical structure. A software module called an XML processor is used to read XML documents and provide access to their content and structure. It is assumed that an XML processor is doing its work on behalf of another module, called the application...."
- **Open Internet Standard.** XML is gaining wide industry support from other vendors besides, like IBM, Sun, Microsoft, Netscape, SAP, CISCO and others, as a platform- and application-neutral format for exchanging information.

Although this manual is not intended to expound on XML syntax, a brief overview of some key XML topics is presented here. You can refer to the many excellent resources listed in "Additional XML Resources" for more information on XML syntax.

How XML Differs From HTML

Like HTML, XML is a subset of SGML (Structured Generalized Markup Language), optimized for delivery over the web.

Unlike HTML, which tags elements in web pages for presentation by a browser, e.g. <bold>Oracle</bold>, XML tags elements as data, e.g. <company>Oracle</company>. For example, you can use XML to give context to words and values in web pages, identifying them as data instead of simple textual or numeric elements.

The following example is in HTML code. This is followed by the corresponding XML example. The examples show employee data:

- Employee number
- Name
- Job
- Salary

HTML Example 1

```
<table>
   <tr><td>EMPNO</td><td>ENAME</td><td>JOB</td><td>SAL</td></tr>
   <tr><td>7654</td><td>MARTIN</td><td>SALESMAN</td><td>1250</td></tr>
   <tr><td>7788</td><td>SCOTT</td><td>ANALYST</td><td>3000</td></tr>
 </table>
```

XML Example 1

In the XML code, note the addition of XML data tags and the nested structure of the elements.

```
<?xml version="1.0"?>
  <EMPLIST>
    <EMP>
    <EMPNO>7654</EMPNO>
    <ENAME>MARTIN</ENAME>
     <JOB>SALESMAN</JOB>
     <SAL>1250</SAL>
    </EMP>
    <EMP>
    <EMPNO>7788</EMPNO>
    <ENAME>SCOTT</ENAME>
    <JOB>ANALYST</JOB>
    <SAL>3000</SAL>
    </EMP>
  </EMPLIST>
```

HTML Example 2

Consider the following HTML that uses tags to present data in a row of a table. Is "Java Programming" the name of a book? A university course? A job skill? You cannot be sure by looking at the data and tags on the page. Imagine a computer program trying to figure this out!

```
<HTML>
  <BODY>
    <TABLE>
     <TR>
     <TD>Java Programming</TD>
     <TD>EECS</TD>
     <TD>Paul Thompson</TD>
     <TD>Ron<BR>Uma<BR>Lindsay</TD>
     </TR>
    </TABLE>
  </BODY>
 </HTML>
```

The analogous XML example has the same data, but the tags indicate what information the data represents, not how it should be displayed. It's clear that "Java Programming" is the Name of a Course, but it says nothing about how it should be displayed.

XML Example 2

```
<?xml version="1.0"?>
  <Course>
      <Name>Java Programming</Name>
    <Department>EECS</Department>
    <Teacher>
      <Name>Paul Thompson</Name>
    </Teacher>
    <Student>
      <Name>Ron</Name>
    </Student>
    <Student>
      <Name>Uma</Name>
    </Student>
    <Student>
      <Name>Lindsay</Name>
    </Student>
  </Course>
```

XML and HTML both represent information:

- XML represents information *content*
- HTML represents the *presentation* of that content

Summary of Differences Between XML and HTML

Figure 24–2 summarizes, how XML differs from HTML.

Table 24–2 XML and HTML Differences

XML	HTML
Represents information *content*	Represents the *presentation* of the content
Has user-defined tags	Has a fixed set of tags defined by standards.
All start tags must have end tags	Current browsers relax this requirement on tags <P>, <B>, and so on.
Attributes must be single or double quoted	Current browsers relax this requirement on tags
Empty elements are clearly indicated	Current browsers relax this requirement on tags
Element names and attributes are case sensitive	Element names and attributes are not case sensitive.

Presenting XML Using Stylesheets

A key advantage of using XML as a datasource is that its *presentation* (such as a web page) can be separate from its *structure* and *content*.

- *Presentation*. Applied stylesheets define its *presentation*. XML data can be presented in various ways, both in appearance and organization, simply by applying different stylesheets.
- *Structure and content:* XML data defines the structure and content.

Stylesheet Uses

Consider these ways of using stylesheets:

- A different interface can be presented to different users based on user profile, browser type, or other criteria by defining a different stylesheet for each presentation style.
- Stylesheets can be used to transform XML data into a format tailored to the specific application that receives and processes the data.

Stylesheets can be applied on the server or client side. The XSL-Transformation Processor (XSL-T Processor) transforms one XML format into XML or any other text-based format such as HTML. Oracle XML Parsers all include an XSL-T Processor.

How to apply stylesheets and use the XSL-T Processor is described in the following sections:

- Chapter 17, "Using XML Parser for Java"
- Chapter 11, "Customizing Discoverer 3i Viewer with XSL"

eXtensible Stylesheet Language (XSL)

eXtensible Stylesheet Language (XSL), the stylesheet language of XML is another W3C recommendation. XSL provides for stylesheets that allow you to do the following:

- Transform XML into XML or other text-based formats such as HTML
- Rearrange or filter data
- Convert XML data to XML that conforms with another Document Type Definition (DTD), an important capability for allowing different applications to share data

Cascading Style Sheets (CSS)

Cascading Style Sheets (CSS1), a W3C specification was originally created for use with HTML documents. With CSS you can control the following aspects of your document's appearance:

- Spacing. Element visbility, position, and size
- Colors and background
- Fonts and text

CSS2 was published by W3C in 1998 and includes the following additional features:

- System fonts and colors
- Automatic numbering
- Supports paged media
- Tables and aura stylesheets

'Cascading' here implies that you can apply several stylesheets to any one document. On a web page deploying CSSm, for example, three stylesheets can apply or cascade:

1. User's preferred stylesheet takes precedence
2. Cascading stylesheet
3. Browser stylesheet

Extensibility and Document Type Definitions (DTD)

Another key advantage of XML over HTML is that it leaves the specification of the tags and how they can be used to the user. You construct an XML document by creating your own tags to represent the meaning and structure of your data.

Tags may be defined by using them in an XML document or they may be formally defined in a Document Type Definition (DTD). As your data or application requirements change, you can change or add tags to reflect new data contexts or extend existing ones.

The following is a simple DTD for the previous XML example:

```
<!ELEMENT EMPLIST (EMP)*>
<!ELEMENT EMP (EMPNO, ENAME, JOB, SAL)>
<!ELEMENT EMPNO (#PCDATA)>
<!ELEMENT ENAME (#PCDATA)>
```

```
<!ELEMENT JOB (#PCDATA)>
<!ELEMENT SAL (#PCDATA)>
]>
```

Note: The DOCTYPE declaration is only used when the DTD is embedded in XML code.

Well-Formed and Valid XML Documents

Well-Formed XML Documents

An XML document that conforms to the structural and notational rules of XML is considered *well-formed*. A well-formed XML document does not have to contain or reference a DTD, but rather can implicitly define its data elements and their relationships. Well-formed XML documents must follow these rules:

- Document must start with the XML declaration, <?xml version="1.0">
- All elements must be contained within one root element
- All elements must be nested in a tree structure without overlapping
- All non-empty elements must have start and end tags

Valid XML Documents

Well-formed XML documents that *also* conform to a DTD are considered *valid*. When an XML document containing or referencing a DTD is parsed, the parsing application can verify that the XML conforms to the DTD and is therefore valid, which allows the parsing application to process it with the assurance that all data elements and their content follow rules defined in the DTD.

Why Use XML?

XML, the internet standard for information exchange is useful for the following reasons:

- *Solves Data Interchange Problems*. It facilitates efficient data communication where the data:
  - Is in many different formats and platforms
  - It must be sent to different platforms
  - Must appear in different formats and presentations
  - Must appear on many different end devices

  In short, XML solves application *data interchange* problems. Businesses can now easily communicate with other businesses and workflow components using XML. See Chapters 2 through 20 for more information and examples of how XML solves data interchange problems.

  Web-based applications can be built using XML which helps the interoperation of web, database, networking, and middleware. XML provides a structured format for data transmission.
- *Industry-Specific Data Objects are Being Designed Using XML*. Organizations such as OAG and XML.org are using XML to standardize data objects on a per-industry basis. This will further facilitate business-to-business data interchange.
- *Database-Resident Data is Easily Accessed, Converted, and Stored Using XML*. Large amounts of business data resides in relational and object-relational tables as the database provides excellent data queriability, scalability and availability. This data can be converted from XML format and stored in object-relational and pure relational database structures or generated from them back to XML for further processing.

Other Advantages of Using XML

Other advantages of using XML include the following:

- You can make your own tags
- Many tools support XML
- XML is an Open standard

- XML parsers built according to the Open standard are interoperable parsers and avoid vendor lock-in. XML specifications are widely industry approved.
- In XML the presentation of data is separate from the data's structure and content. It is simple to customize the data's presentation. See "Presenting XML Using Stylesheets" and "Customizing Your Data Presentation".
- Universality -- XML enables the representation of data in a manner that can be self-describing and thus universally used
- Persistence -- Through the materialization of data as an XML document this data can persist while still allowing programmatic access and manipulation.
- Platform and application independence
- Scalability

Additional XML Resources

Here are some additional resources for information about XML:

- *The Oracle XML Handbook*, Ben Chang, Mark Scardina, et.al., Oracle Press
- *Building Oracle XML Applications*, Steve Muench, O'Reilly
- *XML Bible*, Elliotte Rusty Harold, IDG Books Worldwide
- *XML Unleashed, Morrison et al.*, SAMS
- *Building XML Applications*, St.Laurent and Cerami, McGraw-Hill
- *Building Web Sites with XML*, Michael Floyd, Prentice Hall PTR
- *Building Corporate Portals with XML*, Finkelstein and Aiken, McGraw-Hill
- http://www.w3.org/TR lists W3C technical reports
- http://www.w3.org/xml is the W3C XML activity overview page
- http://www.xml.com includes latest industry news about xml
- http://www.xml-cml.org has information about Chemical Markup Language (CML). CML documents can be viewed and edited on the Jumbo browser.
- http://www.loc.gov/ead/ Encoded Archival Description (EAD) information developed for the US Library of Congress.
- http://www.docuverse.com/xlf for information about Extensible Log Format (XLF) a project to convert log files into XML log files to simplify log file administration.

- http://www.w3.org/Math for information about MathML which provides a way of interchanging equations between applications.
- http://www.naa.org Newspaper Association of America classified ads format for easy exchange of classified ads.
- http://www.w3/org/AudioVideo/ for information about Synchronized Multimedia Integration Language (SMIL).
- Oracle is an official sponsor of OASIS. OASIS, http://www.oasis-open.org, is the world's largest independent, non-profit organization dedicated to the standardization of XML applications. It promotes participation from all industry, and brings together both competitors and overlapping standards bodies.

B

Comparing Oracle XML Parsers and Class Generators by Language

This appendix provides a comparison of the Oracle XML Parser and Class Generators by language. The following sections are included in this appendix:

Comparing the Oracle XML Parsers

Table 24–3 compares the features of the Oracle XML parsers according to language.

Table 24–3 Comparing Oracle XML Parsers

Java	C	C++	PL/SQL
Parser, Version 2			
Includes DOM API	Includes DOM API	Includes DOM API	Includes DOM API
Includes SAX API	Includes SAX API	Includes SAX API	N/A
XSL-T Processor	XSL-T Processor	XSL-T Processor	XSL-T Processor
Namespace 1.0 support	Namespace 1.0 support	Namespace 1.0 support	Namespace 1.0 support
XPath 1.0 support	XPath 1.0 support	XPath 1.0 support	XPath 1.0 support
Checks if document is well-formed	Checks if document is well-formed	Checks if document is well-formed	Checks if document is well-formed
Validating and Non-Validating Support	Validating and Non-Validating Support	Validating and Non-Validating Support	Validating and Non-Validating Support
Character Sets (15): 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	Character Sets (15): 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	Character Sets (15): 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	Character Sets (12): 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

Table 24–3 Comparing Oracle XML Parsers (Cont.)

Java	C	C++	PL/SQL
Default Character Set: UTF-8	Default Character Set: UTF-8	Default Character Set: UTF-8	Default Character Set: UTF-8
Operating Systems: All Oracle8*i* platforms	Operating Systems: All Oracle8*i* platforms	Operating Systems: All Oracle8*i* platforms	Operating Systems: All Oracle8*i* platforms
Error recovery until fatal error	N / A	N / A	Error recovery until fatal error

Comparing the Oracle XML Class Generators

Table 24–3 compares the features of the Oracle XML parsers and class generators, according to language.

Table 24–4 Comparing Oracle XML Parsers and Class Generators

Java	C	C++	PL/SQL
Class Generator			
oracle.xml.classgen		xmlcg	
CGDocument CGNode ClassGenerator InvalidContentException	N / A		N / A
Character Sets (8): EBCDIC-CP-US ISO-8859-1 ISO-10646-UCS-2 ISO-10646-UCS-4 Shift_SJIS US-ASCII UTF-8 UTF-16	N / A	Character Sets (8): EBCDIC-CP-US ISO-8859-1 ISO-10646-UCS-2 ISO-10646-UCS-4 Shift_SJIS US-ASCII UTF-8 UTF-16	N / A
Default Character Set: US-ASCII	N / A	Default Character Set: US-ASCII	N / A

C

XDK for Java: Specifications and Cheat Sheets

This Appendix describes the XDK for Java specifications and cheat sheets for each XML component for Java. The cheat sheets list the main APIs, classes and associated methods for each Oracle XML component.

This chapter contains the following sections:

XML Parser for Java Cheat Sheets

Table C–1 lists XML Parser for Java top level classes with a brief description of each.

Table C–1 XML Parser for Java: oracle.xml.parser.v2 Classes

Class Summary	Description
Interfaces	
NSName	This interface provides Namespace support for Element and Attr names
NSResolver	This interface provides support for resolving Namespaces
XMLDocumentHandler	This interface extends the org.xml.sax.DocumentHandler interface.
XMLToken	Basic interface for XMLToken
Classes	
AttrDecl	Holds information about each attribute declared in an attribute list in the Document Type Definition.
Package Oracle.xml.parser.v2	Implements the default behaviour for the XMLDocumentHandler interface.
DOMParser	Implements an eXtensible Markup Language (XML) 1.0 parser according to the World Wide Web Consortium (W3C) recommendation.
DTD	Implements the DOM DocumentType interface and holds the Document Type Definition information for an XML document.
ElementDecl	Represents an element declaration in a DTD.
NodeFactory	Specifies methods to create various nodes of the DOM tree built during parsing.
oraxsl	Provides a command-line interface to applying stylesheets on multiple XML documents.
SAXAttrList	Implements the SAX AttributeList interface and also provides Namespace support.
SAXParser	Implements an eXtensible Markup Language (XML) 1.0 SAX parser according to the World Wide Web Consortium (W3C) recommendation.
XMLAttr	Implements the DOM Attr interface and holds information on each attribute of an element.
XMLCDATA	Implements the DOM CDATASection interface.
XMLComment	Implements the DOM Comment interface.
XMLDocument	Implements the DOM Document interface, represents an entire XML document and serves the root of the Document Object Model tree.

Table C–1 XML Parser for Java: oracle.xml.parser.v2 Classes

Class Summary	Description
XMLDocumentFragment	Implements the DOM DocumentFragment interface.
XMLElement	Implements the DOM Element interface.
XMLEntityReference	
XMLNode	Implements the DOM Node interface and serves as the primary datatype for the entire Document Object Model.
XMLParser	Serves as a base class for the DOMParser and SAXParser classes.
XMLPI	Implements the DOM Processing Instruction interface.
XMLText	Implements the DOM Text interface.
XMLTokenizer	Implements an eXtensible Markup Language (XML) 1.0 parser according to the World Wide Web Consortium (W3C) recommendation.
XSLProcessor	Provides methods to transform an input XML document using a previously constructed XSLStylesheet.
XSLStylesheet	Holds XSL stylesheet information such as templates, keys, variables, and attribute sets.
Exceptions	
XMLParseException	Indicates that a parsing exception occurred while processing an XML document
XSLException	Indicates that an exception occurred during XSL tranformation

Table C–2 XML Parser for Java: DOMParser() Methods

Method	Description
DOMParser()	Creates a new parser object.
Methods	
getDoctype()	Get the DTD
getDocument()	Gets the document
parseDTD(InputSource, String)	Parses the XML External DTD from given input source
parseDTD(InputStream, String)	Parses the XML External DTD from given input stream.
parseDTD(Reader, String)	Parses the XML External DTD from given input stream.

Table C–2 XML Parser for Java: DOMParser() Methods

Method	Description
parseDTD(String, String)	Parses the XML External DTD from the URL indicated
parseDTD(URL, String)	Parses the XML External DTD document pointed to by the given URL and creates the corresponding XML document hierarchy.
setErrorStream(OutputStream)	Creates an output stream for the output of errors and warnings.
setErrorStream(OutputStream, String)	Creates an output stream for the output of errors and warnings.
setErrorStream(PrintWriter)	Creates an output stream for the output of errors and warnings.
setNodeFactory(NodeFactory)	Set the node factory.
showWarnings(boolean)	Switch to determine whether to print warnings

oraxsl Command Line Interface

The oraxsl class provides a command-line interface to applying stylesheets on multiple XML documents. It accepts a number of command-line options that dictate how it should behave. The following is its invocation syntax:

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

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

     java oraxsl options* source? stylesheet? result?
          -w                        Show warnings
          -e <error log>            A file to write errors to
          -l <xml file list>        List of files to transform
          -d <directory>            Directory with files to transform
          -x <source extension>     Extensions to exclude
          -i <source extension>     Extensions to include
          -s <stylesheet>           Stylesheet to use
          -r <result extension>     Extension to use for results
          -o <result extension>     Directory to place results
          -p <param list>           List of Params
          -t <# of threads>         Number of threads to use
          -v                              Verbose mode
```

Accessing XML Parser for Java

The Oracle XML Parsers are provided with Oracle8i Enterprise and Standard editions from release 8.1.6 and higher. A .... is also available with OracleLite ??? CHECK

If you do not have these editions you can download the XML Parsers from:

http://technet.oracle.com/tech/xml/

Installing XML Parser for Java, Version 2

These sections describe how to install the Windows NT and UNIX versions of the XML Parser for Java, Version 2.

XML Parser for Java, Version 2: Windows NT Installation To install the Oracle XML Parser for Java (v2) on Windows NT follow these steps:

1. Install JDK-1.1.x. or above and either unzip or WinZip executable.
2. Download the Oracle XML Parser in ZIP format.
3. Unzip xmlparser.zip into a directory. For example:

   ```
   C:\[your directory]> unzip xmlparser.zip
   ```

4. The result should be the following files and directories:
   - license.html — copy of license agreement
   - readme.html — release and installation notes
   - doc\ — directory for documents
   - lib\ — directory for parser class files
   - sample\ — sample code files

XML Parser for Java, Version 2: UNIX Installation To install the XML Parser for Java (v2) in UNIX follow these steps:

1. Install JDK-1.1.x or above and GNU gzip.
2. Download the Oracle XML Parser in .tar.gz format.
3. Extract the distribution package into a directory. For example:

   ```
   #gzip -dc xmlparser.tar.gz | tar xvf -
   ```

4. The result should be the following files and directories:
   * license.html — copy of license agreement
   * readme.html — release and installation notes
   * doc/ — directory for documents
   * lib/ — directory for parser class files
   * sample/ — sample code files

Sample Code

See Chapter 17, "Using XML Parser for Java", for sample code and suggestions on how to use the XML Parsers.

XML Parser for Java, Version 2 Specifications

The Oracle XML Parser for Java, Version 2 specifications follow:

- New high performance architecture
- Integrated support for W3C XSLT Final Working Draft
- Supports validation and non-validation modes
- Built-in Error Recovery until fatal error
- Intergrated Document Object Model (DOM) Level 1.0 API
- Integrated SAX 1.0 API
- Supports W3C Recomendation for XML Namespaces

Requirements

Operating Systems: Any with Java 1.1.x support

AVA: JDK 1.1.x. or above.

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.

Online Documentation

Important note: 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 XSL-T processor for transforming XML documents.

Standards Conformance

The parser conforms to the following standards:

The W3C recommendation for Extensible Markup Language (XML) 1.0 at http://www.w3.org/TR/1998/REC-xml-19980210

The W3C recommendation for Document Object Model Level 1 1.0 at http://www.w3.org/TR/REC-DOM-Level-1/

The W3C recommendation for Namespaces in XML at http://www.w3.org/TR/REC-xml-names/

The Simple API for XML (SAX) 1.0 at http://www.megginson.com/SAX/index.html

The W3C final working draft for XSLT at http://www.w3.org/1999/08/WD-xslt

Supported Character Set Encodings

The XML Parser for Java currently supports the following encodings:

- BIG 5
- EBCDIC-CP-*
- EUC-JP
- EUC-KR
- GB2312
- ISO-2022-JP

- ISO-2022-KR
- ISO-8859-1to -9
- ISO-10646-UCS-2
- ISO-10646-UCS-4
- KOI8-R
- Shift_JIS
- US-ASCII
- UTF-8
- UTF-16

Default: UTF-8 is the default encoding if none is specified. Any other ASCII or EBCDIC based encodings that are supported by the JDK may be used.However, they must be specified in the format required by the JDK instead of as official character set names defined by IANA.

Error Recovery

The parser also provides error recovery. It will recover from most errors and continue processing until a fatal error is encountered.

Oracle XML Parser V1 and V2

Version 2 of the XML Parser for Java, besides incorporating an XSL-T processor, has been re-architected from version 1. This has resulted in a number of changes to the class names especially those that support Namespaces. The following summarizes changes you have to take into account when converting code from v1 to v2.

Note: This summary is based upon XML Parser versions v1.0.1.4 as v1 and v2.0.0.0 as v2.

NEW CLASS STRUCTURE

oracle.xml.parser package has been renamed to oracle.xml.parser.v2.

The following are new interfaces:

- NSName

- XMLDocumentHandler

The following interfaces have been removed:

- NSAttr
- NSAttributeList
- NSDocumentHandler
- NSElement

The following are new classes in v2:

- DOMParser
- DefaultXMLDocumentHandler
- SAXAttrList
- SAXParser
- XSLProcessor
- XSLStylesheet
- XSLException

Within this package the classes have been reorganized as follows:

Table C–3 XML Parser for Java: Classes Reoganization and Changes

Version 1	Version 2
Class Reorganization	■
XMLParser	■ XMLParser (includes methods applicable to DOM and SAX access), ■ DOMParser (includes methods specific to DOM access), ■ SAXParser (includes methods specific to SAX access)
NSDocumentHandler	XMLDocumentHandler
NSAttr	XMLAttr (supports Namespace, NSAttr interface has been removed)
NSAttributeList	SAXAttrList (includes methods in NSAttributeList and org.xml.sax.AttributeList)
NSElement	XMLElement (supports Namespace, NSElelement interface removed)

Table C–3 XML Parser for Java: Classes Reoganization and Changes

Version 1	Version 2
PUBLIC CLASS / VARIABLE / CONSTRUCTOR / METHOD CHANGES	
AttrDecl	
getName()	<eliminated> Use XMLNode.getNodeName
getPresence()	getAttrPresence()
getType()	getAttrType()
getValues()	getEnumerationValues()
ElementDecl	
ASTERISK	Rserved for future implementation.
COMMA	Rserved for future implementation.
ELEMENT	Rserved for future implementation.
OR	Rserved for future implementation.
PLUS	Rserved for future implementation.
QMARK	Rserved for future implementation.
getParseTree()	Rserved for future implementation.
XMLAttr	New constructor - XMLAttr(String, String, String, String) New methods - cloneNode(), getPrefix()
XMLElement	New constructor - XMLElement(String, String, String) New methods: ■ checkNamespace(String, String) ■ getElementsByTagName(String, String), ■ resolveNamespacePrefix(String)
XMLNode	New method - transformNode()
XMLText	New method - getNodeValue()

XML Parser for Java Release History

Table C–4 lists the XML Parser for Java release history:

Table C–4 XML Parser for Java V2: Release History

Release	Description
Oracle XML Parser 2.0.2.?	New API in Parser class: ■ reset() : Reset the internal DOM data structures New API in XMLDocument class: ■ print(PrintDriver): Customization of the print functionality New Interface: ■ PrintDriver : Interface for customizing print ■ XMLPrintDriver class : Default implementation of PrintDriver
2.0.2.?	Changes: New builtin extension element: ■ ora:output, where xmlns:ora="http://www.oracle.com/XSL/Transform/java". This element can be used a top-level element similar to xsl:output (can have all attribute allowed in xsl:output, and has similar functionality). It has an additional attribute 'name' used an an identifier. When ora:output is used in a template, it can have only two attribute 'use' and 'href'. 'use' attribute specifies the top-level ora:output to be used, and 'href' gives the output URL. New builtin extension function: ■ ora:node-set, where xmlns:ora="http://www.oracle.com/XSL/Transform/java" ora:node-set function converts a result tree fragment into a node-set.
2.0.2.7.0	Changes: ■ New command line interface to parse XML documents has been added to bin directory. See bin/readme.html for details. ■ Debugging support has been added to XSL-T processing. The errors / warnings include the system identifier, line number, and column number in debug mode. You can either use DOMParser.setDebugMode(boolean) or use Java system property 'oracle.xml.parser.debugmode' to set the debug mode.

Table C–4 XML Parser for Java V2: Release History (Cont.)

Release	Description
2.0.2.7.0 / ...	New API in DOMParser class: ■ setDebugMode(boolean flag) New API in XMLDocument class: ■ setDoctype(String rootname, String sysid, String pubid) - Adds DOCTYPE declaration to the XML Dcoument.
2.0.2.6	Changes: ■ Conformance to the XSLT / XPATH October REC. New API in XSLStylesheet class: ■ removeParam(String param) ■ resetParams()
2.0.2.5	Changes: ■ Conformance to the XSLT / XPATH October PR. ■ Support for internationalized error messages has been added. The locale can be set using setLocale(java.util.Locale) function in XSLProcessor, SAXParser, and DOMParser. The following locales are supported:

Country	Language	Country Code	Language Code
ARGENTINA	SPANISH	AR	es
BRAZIL	BRAZILIAN PORTUGUESE	BR	pt
CHINA	CHINESE	CN	zh
CZECH REPUBLIC	CZECH	CZ	cs
DENMARK	DANISH	DK	da
FINLAND	FINNISH	FI	fi
FRANCE	FRENCH	FR	fr
GERMANY	GERMAN	DE	de
GREECE	GREEK	GR	el
HUNGARY	HUNGARIAN	HU	hu
ISRAEL	HEBREW	IL	iw
ITALY	ITALIAN	IT	it
JAPAN	JAPANESE	JP	ja

Table C–4 XML Parser for Java V2: Release History (Cont.)

Release	Description			
	Country	**Language**	**Country Code**	**Language Code**
	KOREA	KOREAN	KR	ko
	NORWAY	NORWEGIAN	NO	no
	POLAND	POLISH	PL	pl
	PORTUGAL	PORTUGUESE	PT	pt
	ROMANIA	ROMANIAN	RO	ro
	RUSSIA	RUSSIAN	RU	ru
	SLOVAKIA	SLOVAK	SK	sk
	SPAIN	CATALAN	ES	ca
	SPAIN	SPANISH	ES	es
	SWEDEN	SWEDISH	SE	sv
	TAIWAN	CHINESE	TW	zh
	THE NETHERLANDS	DUTCH	NL	nl
	TURKEY	TURKISH	TR	tr
	UNITED ARAB EMIRATES	ARABIC	AE	ar
	UNITED STATES	ENGLISH	US	en
	New APIs in XMLNode class: ■ value-of(String pattern) ■ selectNodes(String pattern) ■ selectSingleNode(String pattern) ■ selectSingleNode(String pattern, NSResolver ns) New API in XSLStylesheet class ■ setParam(String param, String value)			
2.0.2.4.0	XSL URI constant is now: http://www.w3.org/1999/XSL/Transform Extension functions namespace has changed from http://www.oracle.com/oracle.xml.parser/xsl/java/classname to http://www.oracle.com/XSL/Transform/java/classname			
2.0.2.3.0	Extension functions support was added, that allows users of the XSL processor to call any Java method from XSL expressions. Please refer to extfunc.html for more details on Extension functions.			

Table C–4 XML Parser for Java V2: Release History (Cont.)

Release	Description
2.0.2.2.0	New API in XSLProcessor class to support xsl:output element: ■ processXSL(XSLStylesheet xsl, XMLDocument xml, OutputStream out) ■ processXSL(XSLStylesheet xsl, XMLDocument xml, PrintWriter pw) ■ processXSL(XSLStylesheet xsl, XMLDocumentFragment xml, OutputStream out) ■ processXSL(XSLStylesheet xsl, XMLDocumentFragment xml, PrintWriter pw)
2.0.2.1.0	Changes: ■ Bug#992758: Fix in invalid character detection. ■ Bug#995394: Fix in forward referencing of IDs. ■ Bug fix in XPath Boolean and Number expression parsing. ■ Bug fix in XSL:Number with empty count attribute. This is the first production patch release for v2.
2.0.2.0.0	Changes: Conformance to the XSLT/XPATH August WD. **Note**: There are several changes between April99 XSL-T draft and the August99 XSL-T/Xpath draft and these changes have been implemented in the XSL Processor. The XSL Processor has been modified to accept XPath syntax for expressions and patterns. Stylesheets might have to be modified to be conformant to the August XSL-T/XPath draft before they can be used with this release. Some of the changes between the April and August draft are: ■ Expressions in the stylesheet must match the XPath production Expr. ■ Some of the attribute names and element names in XSL namespace have changed. ■ Some new functions have been added to XPath CORE function library. Refer to the August XSL-T/XPath draft for more details. This is the first production release for v2.

Table C–4 XML Parser for Java V2: Release History (Cont.)

Release	Description
	Changes: Note the command line version of 'oraxsl' which can be used in scripts and can specify multiple # of threads has been added to the bin directory. Note that the 'xmlparser.jar' file name has been changed to 'xmlparserv2.jar' to allow co-existence with v1 Java XML parser. Bug fixes for #972862, i.e., DOM exception when the DTD is cached rather than loaded everytime for multiple files; #968640, i.e., support for character set encoding names that are non-IANA; #929682, i.e., a command-line version of XSLT is now bundled (see the bin directory); #978320, i.e., error adding nodes to a cloned document using CloneNode() method; for problems with attribute validation / setDTD / parseDTD. Improved Lexer. With JDK 1.6, the parser seems to have better performance overall because of changes in the lexical analyzer. New Interface: XMLToken. Applications can register an interface handle and the interface callback method 'token()' receives XML tokens registered and found by the parser. XML tokens are defined as interface constants. They are: STagName, EmptyElemTag, STag, ETag, ETagName, Attribute, AttName, AttValue, Reference, Comment, CharData, CDSect,
	PI, PITarge, XMLDecl, TextDecl,

Table C–4 XML Parser for Java V2: Release History (Cont.)

Release	Description
	DTDName, AttListDecl, elementdecl, ElemDeclName, EntityDecl, EntityDeclName, EntityValue, NotationDecl, ExternalID.
	The tokens above correspond to the syntax variables from W3C XML specifications. New API in the following Classes: 1. oracle.xml.parser. -setTokenHandler (XMLToken handler). Sets XMLtoken interface handler. -setToken (int token, boolean val). Registers on / off XML token.

Table C–4 XML Parser for Java V2: Release History (Cont.)

Release	Description
2.0.0.1.0	Changes: Bug fixes for #920536, i.e. Cannot access element attributes via XSLT; #898423. i.e. ElementDecl's in DTDs; #774774, i.e. DOM extensions using XSL pattern matching; #863890 i.e. SAX IOException not thrown. API in the following new interface: 1. oracle.xml.parser.v2.NSResolver ■ resolveNamespacePrefix(find the namespace definition in scope for a given namespace prefix) New APIs in the follo- selectNodes(Selects nodes from the tree which match the given pattern; client can provide an NSResolver implementation to resolve namespace prefixes in the pattern). 2. oracle.xml.parser.v2.ElementDecl ■ getParseTree(Returns the root Node of Content Model parse tree, which could then be traversed node by node using getFirstChild() and getLastChild(). The Node types are: PLUS, COMMA, ASTERISK, ELEMENT, QMARK). This is the first beta patch release for v2. wing classes: ???? 1. oracle.xml.parser.v2.XMLNode
2.0.0.0.0	The Oracle XML v2 parser is an early beta release and is written in Java. It features an improved architecture over the Oracle XML V1 parser and has shown better performance on small to large XML documents. It will also be able to format the XML document according to a stylesheet, having integrated an XSLT processor. **Note**: Several of the package names and error messages are different than in the v1 of the parser. See below and v2changes.txt and the .diff difference files in the sample directory. Version 2 of the XML Parser for Java, besides incorporating an XSLT processor, has been re-architected from version 1. This has resulted in a number of changes to the class names especially those that support Namespaces. The following is a summary:

Table C–4 XML Parser for Java V2: Release History (Cont.)

Release	Description
	NOTE: This summary is based upon XML Parser versions v1.0.1.4 as v1 and v2.0.0.0 as v2. NEW CLASS STRUCTURE The oracle.xml.parser package has been renamed to oracle.xml.parser.v2. The following are the new interfaces: ■ NSName ■ XMLDocumentHandler ■ The following interfaces have been removed: ■ NSAttr ■ NSAttributeList ■ NSDocumentHandler ■ NSElement ■ The following are the new classes in v2: ■ DOMParser ■ DefaultXMLDocumentHandler ■ SAXAttrList ■ SAXParser ■ XSLProcessor ■ XSLStylesheet ■ XSLException

XDK for Java: XML Java Class Generator

Installing XML Java Class Generator

Installing the Oracle XML Java Class Generator, is described in the following sections.

XML Java Class Generator: Windows NT Installation

To install Oracle XML Class Generators on Windows NT, follow these steps:

4. Install JDK-1.1.x. or above and either unzip or run the WinZip executable.
5. Download the Oracle XML Class Generator for Java in ZIP format from http://technet.oracle.com/tech/xml/classgen

   This Class Generator for Java uses 54Kb. Select the following:

   - Software, from the XML top menu >
   - Oracle XML Class Generator for Windows NT
6. Unzip xmlclassgenV1_0_0.zip into a directory. For example:

   ```
   C:\[your directory]>unzip xmlclassgenV1_0_0.zip
   ```

7. The result should be the following files and directories:
   - license.html — copy of license agreement
   - readme.html — release and installation notes
   - doc\ — directory for documents
   - lib\ — directory for classgen class files
   - sample\ — sample code files

XML Java Class Generator: UNIX Installation

To install Oracle XML Class Generator for Java in UNIX, follow these steps:

1. Install JDK-1.1.x or above and GNU gzip.
2. Download the Oracle XML Class Generator for Java in .tar.gz format from

   http://technet.oracle.com/tech/xml/classgen

   This Class Generator for Java uses 41Kb. Select the following:

- Software, from the XML top menu >
- Oracle XML Class Generator for UNIX

3. Extract the distribution package into a directory. For example:

```
#gzip -dc xmlclassgenV1_0_0.tar.gz | tar xvf -
```

4. The result should be the following files and directories:
   - license.html — copy of license agreement
   - readme.html — release and installation notes
   - doc/ — directory for documents
   - lib/ — directory for classgen class files
   - sample/ — sample code files

XML Java Class Generator Cheat Sheet

Table C–5 lists the main XML Parser for Java APIs and top level classes with a brief description of each.

Table C–5 XML Java Class Generator: APIs and Classes

Classes	Description
Classes	
CGDocument	Serves as the base document class for the Class Generated generated classes
CGNode	Serves as the base class for nodes generated by the Class Generated
ClassGenerator	This class is used by the Class Generated to generate classes
Exceptions	
InvalidContentException	Definition of InvalidContentException thrown by dtdcompiler classes

Table C–6 XML Java Class Generator: Default Constructors

ClassGenerator()	**Default constructor for ClassGenerator.**
generate(DTD, String) Class	Traverses the DTD with element doctype as root and generates Java classes

Table C–6 XML Java Class Generator: Default Constructors(Cont.)

ClassGenerator()	**Default constructor for ClassGenerator.**
setGenerateComments(boolean)	Switch to determine whether to generate java doc comments. Default - TRUE
setJavaPackage(String)	Sets the package for the classes generated. Default - No package
setOutputDirectory(String)	Sets the output directory. Default - current directory
setSerializationMode(boolean)	Switch to determine if the DTD should be saved as a serialized object or as text file.
setValidationMode(boolean)	Switch to determine whether the classes generated should validate the XML Document being constructed.Default - TRUE
CGDocument(String, DTD) Class	Constructor for the Root element of the DTD. public abstract class CGDocument extends CGNode. Serves as the base document class for the DTD compiler generated classes
print(OutputStream)	Prints the constructed XML Document
print(OutputStream, String)	Prints the constructed XML Document
public abstract class CGNode	extends Object. Serves as the base class for nodes generated by the XML Class Generator.
CGNode(String)	Constructor for the Elements of the DOM Tree
addData(String)	Adds PCDATA to the Element
addNode(CGNode)	Adds a node as a child to the element
getCGDocument()	Gets the base document (root Element)
getDTDNode()	Gets the static DTD from the base document
setAttribute(String, String)	Sets the value of the Attribute
setDocument(CGDocument)	Sets the base document (root Element)
storeID(String, String)	Store this value for an ID identifier, so that we can later verify IDREF values
storeIDREF(String, String)	Store this value for an IDREF identifier, so that we can later verify, if an corresponding ID was defined.

Table C–6 XML Java Class Generator: Default Constructors(Cont.)

ClassGenerator()	**Default constructor for ClassGenerator.**
validateContent()	Checks if the content of the element is valid as per the Content Model specified in DTD
validEntity(String)	Checks if the ENTITY identifier is valid
validID(String)	Checks if the ID identifier is valid
validNMTOKEN(String)	Checks if the NMTOKEN identifier is valid

XDK for Java: Transviewer Bean Cheat Sheet

Table C–7 lists the main XDK for Java: Transviewer Beans APIs and top level classes with a brief description of each.

Table C–7 XDK for Java: Transviewer Bean APIs and Classes

Class Summary	Description
Interfaces	
DOMBuilderErrorListener	This interface must be implemented in order to receive notifications when error is found during parsing.
DOMBuilderListener	This interface must be implemented in order to receive notifications about events during the asyncronous parsing.
XSLTransformerErrorListener	This interface must be implemented in order to receive notifications about error events during the asyncronous transformation.
XSLTransformerListener	This interface must be implemented in order to receive notifications about events during the asyncronous transformation.
Classes	
DOMBuilder	This class implements an eXtensible Markup Language (XML) 1.0 parser according to the World Wide Web Consortium (W3C) recommendation.
DOMBuilderBeanInfo	
DOMBuilderErrorEvent	This class defines the error event which is sent when parse exception occurs.
DOMBuilderEvent	The event object that DOMBuilder uses to notify all registered listeners about parse events.
ResourceManager	Simple semaphore that maintains access to fixed number of logical resources.
XSLTransformer	Applies XSL transformation in a background thread.
XSLTransformerBeanInfo	
XSLTransformerErrorEvent	The error event object that XSLTransformer uses to notify all registered listeners about transformation error events.
XSLTransformerEvent	The event object that XSLTransformer uses to notify all registered listeners about transformation events.

Table C–8 async.XSLTransformer API

Member Summary	
Constructors	
XSLTransformer()	XSLTransformet constructor
XSLTransformer(int)	XSLTransformer constructor
Methods	
addXSLTransformerErrorListener(XSLTransformerErrorListener)	Adds an error event listener
addXSLTransformerListener(XSLTransformerListener)	Adds a listener
	Returns the unique XSLTransformer id
getResult()	Returns the document fragment for the resulting document.
processXSL(XSLStylesheet, InputStream, URL)	Initiate XSL Transformation in the background.
processXSL(XSLStylesheet, Reader, URL)	Initiate XSL Transformation in the background.
processXSL(XSLStylesheet, URL, URL)	Initiate XSL Transformation in the background.
	Initiate XSL Transformation in the background.
processXSL(XSLStylesheet, XMLDocument, OutputStream)	Initiate XSL Transformation in the background.
removeDOMTransformerErrorListener(XSLTransformerErrorListener)	Renmoves an error event listener
removeXSLTransformerListener(XSLTransformerListener)	Removes a listener
run()	
	Sets the error stream used by the XSL processor
showWarnings(boolean)	Sets the showWarnings flage used by the XSL processor

XDK for Java: XSQL Servlet

Installing Oracle XSQL Servlet

Downloading and Installing XSQL Servlet

Downloading XSQL Servlet from Oracle Technet

You can download XSQL Servlet distribution from:

http://technet.oracle.com/tech/xml/xsql_servlet

1. Click on the 'Software' icon at the top of the page:
2. Log in with your OTN username and password (registration is free if you do not already have an account).
3. Selecting whether you want the NT or Unix download (both contain the same files)
4. Acknowledge the licensing agreement and download survey
5. Clicking on xsqlservlet_v0_9_9_1.tar.gz or xsqlservlet_v0_9_9_1.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_v0_9_9_1.tar.gz
   ```

   Windows NT:

   ```
   pkzip25 -extract -directories xsqlservlet_v0_9_9_1.zip
   ```

   using the pkzip25 command-line tool or the WinZip visual archive extraction tool.

Windows NT: Starting the Web-to-go Server

XSQL Servlet comes bundled with the Oracle Web-to-go server that is pre-configured to use XSQL Pages. The Web-to-go web server is a single-user server, supporting the Servlet 2.1 API, used for mobile application deployment and for development. This is a great way to try XSQL Pages out on your Windows machine before delving into the details of configuring another Servlet Engine to run XSQL Pages.

Note: The Web-to-go Web server is part of Oracle's development and deployment platform for mobile applications. For more information on Web-to-go, see http://www.oracle.com/mobile.

Windows NT users can get started quickly with XSQL Pages by following these steps:

1. Running the xsql-wtg.bat script in the .\xsql directory.
2. Browsing the URL http://localhost:7070/xsql/index.html

If you get an error starting this script, edit the xsql-wtg.bat file to properly set the two environment variables JAVA and XSQL_HOME to appropriate values for your machine.

```
REM ------------------------------------------------
REM Set the 'JAVA' variable equal to the full path
REM of your Java executable.
REM ------------------------------------------------
set JAVA=J:\java1.2\jre\bin\java.exe
set XSQL_HOME=C:\xsql
REM ------------------------------------------------
REM Set the 'XSQL_HOME' variable equal to the full
REM path of where you install the XSQL Servlet
REM distribution.
REM ------------------------------------------------
```

Then, repeat the two steps above.

If you get an error connecting to the database when you try the demos, you'll need to go on to the next section, then try the steps above again after setting up your database connection information correctly in the XSQLConfig.xml file.

Setting Up the Database Connection Definitions for Your Environment

The demos are set up to use the SCOTT schema on a database on your local machine (i.e. 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:

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_v0_9_9_1.tar.gz and xsqlservlet_v0_9_9_1.zipdistributions 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.

1. Map the .xsql file extension to the oracle.xml.xsql.XSQLServlet servlet class

2. 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 XSL-T
- Supports W3C XML 1.0 Recommendation
- Supports Document Object Model (DOM) Level 1.0 API
- Support the W3C XSLT Final Working Draft
- Supports W3C Recomendation for XML Namespaces

Character Set Support

XSQL Servlet supports the following character set encodings:

- BIG
- EBCDIC-CP-*
- EUC-JP
- EUC-KR
- GB2312
- ISO-2022-JP
- ISO-2022-KR
- ISO-8859-1to -9
- ISO-10646-UCS-2
- ISO-10646-UCS-4
- KOI8-R
- Shift_JIS
- US-ASCII
- UTF-8

- UTF-16

XDK for Java: XSQL Servlet Cheat Sheets

Table C–9 and Table C–10 list XSQL Servlet APIs and top level classes with a brief description of each.

Table C–9 XSQL Servlet Classes

Class Summary	Description
Interfaces	
XSQLActionHandler	Interface that must be implemented by all XSQL Action Element Handlers
XSQLPageRequest	Interface representing a request for an XSQL Page
Classes	
Res	
XSQLActionHandlerImpl	Base Implementation of XSQLActionHandler that can be extended to create your own custom handlers.
XSQLCommandLine	Command-line Utility to process XSQL Pages.
XSQLDiagnostic	
XSQLHttpUtil	
XSQLPageRequestImpl	Base implementation of the XSQLPageRequest interface that case be used to derive new kinds of page request implementations.
XSQLParserHelper	Common XML Parsing Routines
XSQLRequest	Programmatically process a request for an XSQL Page.
XSQLServlet	Servlet to enable HTTP GET-ing of and POST-ing to XSQL Pages
XSQLServletPageRequest	Implementation of XSQLPageRequest for Servlet-based XSQL Page requests.
XSQLStylesheetProcessor	XSLT Stylesheet Processing Engine
XSQLUtil	
Class Summary	
Interfaces	

Table C–9 XSQL Servlet Classes (Cont.)

Class Summary	Description
XSQLActionHandler	Interface that must be implemented by all XSQL Action Element Handlers
XSQLPageRequest	Interface representing a request for an XSQL Page
Classes	
Res	
XSQLActionHandlerImpl	Base Implementation of XSQLActionHandler that can be extended to create your own custom handlers.
XSQLCommandLine	Command-line Utility to process XSQL Pages.
XSQLDiagnostic	
XSQLHttpUtil	
XSQLPageRequestImpl	Base implementation of the XSQLPageRequest interface that case be used to derive new kinds of page request implementations.
XSQLParserHelper	Common XML Parsing Routines
XSQLRequest	Programmatically process a request for an XSQL Page.
XSQLServlet	Servlet to enable HTTP GET-ing of and POST-ing to XSQL Pages
XSQLServletPageRequest	Implementation of XSQLPageRequest for Servlet-based XSQL Page requests.
XSQLStylesheetProcessor	XSLT Stylesheet Processing Engine
XSQLUtil	

Table C–10 XSQL Servlet: XSQLPageRequest() Class

Methods	Description
createNestedRequest(URL, Dictionary)	Returns an instance of a nested Request
getConnectionName()	Returns the name of the connection being used for this request May be null if no connection set / in-use.
getErrorWriter()	Returns a PrintWriter to print out errors processing this request

Table C–10 XSQL Servlet: XSQLPageRequest() Class (Cont.)

getJDBCConnection()	Gets the JDBC connection being used for this request (can be null)
getPageEncoding()	Returns encoding of source XSQL Page associated with this request
getParameter(String)	Returns the value of the requested parameter
getPostedDocument()	Returns the content of Posted XML for this request as an XML Document
getRequestParamsAsXMLDocument()	Returns the content of a Request parameters as an XML Document
getRequestType()	Returns a string identifying the type of page request being made.
getSourceDocumentURI()	Returns a String representation of the requested document's URI
getStylesheetParameter(String)	Gets a stylesheet parameter by name
getStylesheetParameters()	Gets an enumeration of stylesheet parameter names
getStylesheetURI()	Returns the URI of the stylesheet to be used to process the result.
getUserAgent()	Returns a String identifier of the requesting program
getWriter()	Returns a PrintWriter used for writing out the results of a page request
getXSQLConnection()	Gets the XSQLConnection Object being used for this request Might be null.
isIncludedRequest()	Returns true if this request is being included in another.
isOracleDriver()	Returns true if the current connection uses the Oracle JDBC Driver
printedErrorHeader()	Returns the state of whether an Error Header has been printed
requestProcessed()	Allows Page Request to Perform end-of-request processing
setConnectionName(String)	Sets the connection name to use for this request
setContentType(String)	Sets the content type of the resulting page
setIncludingRequest(XSQLPageRequest)	Sets the Including Page Request object for this request.

Table C–10 XSQL Servlet: XSQLPageRequest() Class (Cont.)

setPageEncoding(String)	Sets encoding of source XSQL page associated with this request.
setPageParam(String, String)	Sets a dynamic page parameter value.
setPostedDocument(Document)	Allows programmatic setting of the Posted Document
setPrintedErrorHeader(boolean)	Sets whether an Error Header has been printed
setStylesheetParameter(String, String)	Sets the value of a parameter to be passed to the associated stylesheet
setStylesheetURI(String)	Sets the URI of the stylesheet to be used to process the result.
translateURL(String)	Returns a string representing an absolute URL resolved relative to the base URI for this request.
useConnectionPooling()	Returns true if connection pooling is desired for this request
useHTMLErrors()	Returns true if HTML-formatted error messages are desired for this request

D

XDK for C: Specifications and Cheat Sheets

This appendix contains the following sections:

XML Parser for C Specifications

Oracle provides a set of XML parsers for Java, C, C++, and PL/SQL. Each of these parsers is a stand-alone XML component that parses an XML document (or a standalone DTD) so that it can be processed by an application. Library and command-line versions are provided and support the following "standards" and features:

- DOM (Document Object Model) support is provided compliant with the W3C DOM 1.0 Recommendation. These APIs permit applications to access and manipulate an XML document as a tree structure in memory. This interface is used by such applications as editors.
- SAX (Simple API for XML) support is also provided compliant with the SAX 1.0 specification. These APIs permit an application to process XML documents using an event-driven model.
- Support is also included for W3C recommendation for XML Namespaces 1.0 thereby avoiding name collisions, increasing reusability and easing application integration.
- Supports validation and non-validation modes.
- Supports W3C XML 1.0 Recommendation.
- Integrated support for W3C XSLT 1.0 Recommendation.

Validating and Non-Validating Mode Support

The XML Parser for C can parse XML in validating or non-validating modes.

- In *non-validating mode*, the parser verifies that the XML is well-formed and parses the data into a tree of objects that can be manipulated by the DOM API.
- In *validating mode*, the parser verifies that the XML is well-formed and validates the XML data against the DTD (if any).

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

Example Code

See Chapter 21, "Using XML Parser for C" for example code and suggestions on how to use the XML Parser for C.

Online Documentation

Documentation for Oracle XML Parser for C is located in the $ORACLE_HOME/xdk/c/parser/doc directory.

Release Specific Notes

The readme.html file in the root directory of the archive contains release specific information including bug fixes, API additions, and so on.

The Oracle XML parser for C is written in C. It will check if an XML document is well-formed, and optionally validate it against a DTD. The parser will construct an object tree which can be accessed via a DOM interface or operate serially via a SAX interface.

Standards Conformance

XML Parser for C conforms to the following standards:

- The W3C recommendation for Extensible Markup Language (XML) 1.0 at http://www.w3.org/TR/1998/REC-xml-19980210
- The W3C recommendation for Document Object Model Level 1 1.0 at http://www.w3.org/TR/REC-DOM-Level-1/
- The W3C proposed recommendation for Namespaces in XML at http://www.w3.org/TR/1998/PR-xml-names-19981117
- The Simple API for XML (SAX) 1.0 at http://www.megginson.com/SAX/index.html
- The W3C Recommendation for XSL Transform 1.0 at http://www.w3.org/TR/xslt

Supported Character Set Encodings

XML Parser for C supports documents in the following encodings, in addition to the ones specified in Appendix A, "Character Sets", of *Oracle8i National Language Support Guide*:

- BIG 5
- EBCDIC-CP-*
- EUC-JP
- EUC-KR

- GB2312
- ISO-2022-JP
- ISO-2022-KR
- ISO-8859-1, ISO-8859-2, ISO-8859-3, ..., ISO-8859-9
- ISO-10646-UCS-2
- ISO-10646-UCS-4
- KOI8-R
- Shift_JIS
- US-ASCII
- UTF-8
- UTF-16

Default: The default encoding is UTF-8. It is recommended that you set the default encoding explicitly if using only single byte character sets (such as US-ASCII or any of the ISO-8859 character sets) for performance up to 25% faster than with multibyte character sets, such as UTF-8.

Release Specific Notes

The readme.html file in the root directory of the archive contains release specific information including bug fixes, API additions, etc..

Oracle XML Parser for C is an early beta release and is written in C. It will check if an XML document is well-formed, and optionally validate it against a DTD. The parser will construct an object tree which can be accessed via a DOM interface or operate serially via a SAX interface. Neither interface is 100% complete in this release.

Standards Conformance

The XML Parser for C 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/

- W3C proposed recommendation for Namespaces in XML at http://www.w3.org/TR/1998/PR-xml-names-19981117
- Simple API for XML (SAX) 1.0 at http://www.megginson.com/SAX/index.html

XML Parser for C Revision History

Table D–1 lists the XML Parser for C revision history.

Table D–1 XML Parser for C: Revision History

Revision	Description
XML Parser 2.0.4.0.0 (C)	This is the first production V2 release. This changes in this release were mainly bug fixes. For the XML parser, the following bugs were fixed: ■ 1352943 XMLPARSE() SOMETIMES CHOKES ON FILENAMES ■ 1302311 PROBLEM WITH PARAMETER ENTITY PROCESSING ■ 1323674 INCONSISTENT ERROR HANDLING IN THE C XML PARSER ■ 1328871 LPXPRINTBUFFER UNCONDITIONALLY PREPENDS XML COMMENT TO OUTPUT ■ 1349962 USING FREED MEMORY LOCATION CAUSES TLPXVNSA31.DIF oraxmldom.h was renamed to oradom.h
	For the XSLT processor, the following bugs were fixed: ■ 1225546 USELESS ERROR MESSAGE NEEDS DETAIL ■ 1267616 TLPXST14.DIF: REPLACE DBL_MAX WITH SBIG_ORAMAXVAL IN LPXXP.C:LPXXPSUBSTRING() ■ 1289228 ERROR CONTEXT REQUIRED FOR DEBUGGING: FILE NAME, LINE#, FUNCTION, ETC
	■ 1289214 XSL:CHOOSE DOESN'T WORK ■ 1298028 XPATH CONSTRUCT NOT(POSITION()=LAST()) NOT WORKING ■ 1298193 XPATH FUNCTIONS DON'T PROVIDE IMPLICIT TYPE CONVERSION OF PARAMS ■ 1323665 C XML PARSER CANNOT SET BASE DIRECTORY OR URI FOR STYLESHEET PARSING ■ 1325452 SEVERE MEMORY CONSUMPTION / LEAK IN XSLPROCESS ■ 1333693 CHAINED TRANSFORMS WITH C XSL PROCESSOR DON'T WORK: LPX-00002

Table D–1 XML Parser for C: Revision History

Revision	Description
XML Parser 2.0.3.0.0 (C)	SAX memory usage: Much smaller, and flat for any input size and multiple parses (memory leaks plugged). XSLT memory usage: Improved. Validation warnings: Validity Constraint (VC) errors have been changed to warnings and do not terminate parsing. For compatibility with the old behavior (halt on warnings as well as errors), a new flag XML_FLAG_STOP_ON_WARNING (or '-W' to the xml program) has been added. Performance improvements: Switch to finite automata VC structure validation yields 10% performance gain. HTTP support: HTTP URIs are now supported; look for FTP in the next release. For other access methods, the user may define their own callbacks with the new xmlaccess() API.
Oracle XML Parser 2.0.2.0.0 (C)	XSLT improvements: Various bugs fixed in the XSLT processor; error messages are improved; xsl:number, xsl:sort, xsl:namespace-alias, xsl:decimal-format, forwards-compatible processing with xsl:version, and literal result element as stylesheet are now available; the following XSLT-specific additions to the core XPath library are now available: current(), format-number(), generate-id(), and system-property(). Bug fixes: Some problems with validation and matching of start and end tags with SAX were fixed (1227096). Also, a bug with parameter entity processing in external entities was fixed (1225219).
Oracle XML Parser 2.0.1.0.0 (C)	Performance improvements: Major performance improvement over the last, about two and a half times faster for UTF-8 parsing and about four times faster for ASCII parsing. Comparison timing against previous version for parsing (DOM) and validating various standalone files (SPARC Ultra 1 CPU time): File size Old UTF-8 New UTF-8 Speedup Old ASCII New ASCII Speedup 42K 180ms 70ms 2.6 120ms 40ms 3.0 134K 510ms 210ms 2.4 450ms 100ms 4.5 247K 980ms 400ms 2.5 690ms 180ms 3.81M 2860ms 1130ms 2.5 1820ms 380ms 4.82 7M 10550ms 4100ms 2.6 7450ms 1930ms 3.9 10.5M 42250ms 16400ms 2.6 29900ms 7800ms 3.8.

Table D–1 XML Parser for C: Revision History

Revision	Description
	Conformance improvements: Stricter conformance to the XML 1.0 spec yields higher scores on standard test suites (Jim Clark, Oasis, ...). Lists, not arrays: Internal parser data structures are now uniformly lists; arrays have been dropped. Therefore, access is now better suited to a firstChild / nextSibling style loop instead of numChildNodes / getChildNode. DTD parsing:A new API call xmlparsedtd() is added which parses an external DTD directly, without needing an enclosing document. Used mainly by the Class Generator.
	Error reporting: Error messages are improved and more specific, with nearly twice as many as before. Error location is now described by a stack of line number / entity pairs, showing the final location of the error and intermediate inclusions (e.g. line X of file, line Y of entity). NOTE: You must use the new error message file (lpxus.msb) provided with this release; the error message file provided with earlier releases is incompatible. See below. XSL improvements: Various bugs fixed in the XSLT processor; xsl:call-template is now fully supported.
Oracle XML Parser 2.0.0.0.0 (C)	Oracle XML v2 parser is a beta release and is written in C. The main difference from the Oracle XML v1 parser is the ability to format the XML document according to a stylesheet via an integrated an XSLT processor. The XML parser will check if an XML document is well-formed, and optionally validate it against a DTD. The parser will construct an object tree which can be accessed via a DOM interface or operate serially via a SAX interface. Supported operating systems are Solaris 2.6, Linux 2.2, HP-UX 11.0, and NT 4 / Service Pack 3 (and above). Be sure to read the licensing agreement before using this product.

XML Parser for C: Parser Functions

Table D–2 lists the XML Parser for C Parser functions, a brief description, and syntax.

Table D–2 XML Parser for C: Parser Function s

Function	Brief Description	Syntax and Comments
xmlinit	Initialize XML parser	xmlctx *xmlinit (uword *err, const oratext *encoding, void (*msghdlr)(void *msgctx, const oratext *msg, ub4 errcode), void *msgctx, const xmlsaxcb *saxcb, void *saxcbctx, const xmlmemcb *memcb, void *memcbctx, const oratext *lang);
xmlclean	Clean up memory used during parse	void xmlclean(xmlctx *ctx); For those who want to parse multiple files but would like to free the memory used for parses before the subsequent call to xmlparse() or xmlparsebuf().
xmlparse	Parse a file	uword xmlparse(xmlctx *ctx, const oratext *filename, const oratext *encoding, ub4 flags); Flag bits must be OR'd to override the default behavior of the parser. The following flag bits may be set: ■ XML_FLAG_VALIDATE turns validation on. ■ XML_FLAG_DISCARD_WHITESPACE discards whitespace where it appears to be insignificant. The default behavior is to not validate the input. The default behavior for whitespace processing is to be fully conformant to the XML 1.0 spec, i.e. all whitespace is reported back to the application but it is indicated which whitespace is ignorable.
xmlparsebuf	Parse a buffer	uword xmlparsebuf(xmlctx *ctx, const oratext *buffer, size_t len, const oratext *encoding, ub4 flags);
xmlterm	Shut down XML parser	uword xmlterm(xmlctx *ctx);
createDocument	Create a new document	xmlnode* createDocument(xmlctx *ctx) An XML document is always rooted in a node of type DOCUMENT_NODE-- this function creates that root node and sets it in the context.
isStandalone	Return document's standalone flag	boolean isStandalone(xmlctx *ctx) Returns the boolean value of the document's standalone flag, as specified in the <?xml?> processing instruction.

XML Parser for C: DOM API Functions

Table D–3 lists the XML Parser for C DOM API functions.

Table D–3 XML Parser for C: DOM API Functions

Function	Brief Description
appendChild	Append child node to current node
appendData	Append character data to end of node's current data
cloneNode	Create a new node identical to the current one
createAttribute	Create an new attribute for an element node
createCDATASection	Create a CDATA_SECTION node
createComment	Create a COMMENT node
createDocumentFragment	Create a DOCUMENT_FRAGMENT node
createElement	Create an ELEMENT node
createEntityReference	Create an ENTITY_REFERENCE node
createProcessingInstruction	Create a PROCESSING_INSTRUCTION (PI) node
createTextNode	Create a TEXT node
deleteData	Remove substring from a node's character data
getAttrName	Return an attribute's name
getAttrSpecified	Return value of attribute's specified flag [DOM getSpecified]
getAttrValue	Return the value of an attribute
getAttribute	Return the value of an attribute
getAttributeIndex	Return an element's attribute given its index
getAttributeNode	Get an element's attribute node given its name [DOM getName]
getAttributes	Return array of element's attributes
getCharData	Return character data for a TEXT node [DOM getData]
getCharLength	Return length of TEXT node's character data [DOM getLength]
getChildNode	Return indexed node from array of nodes [DOM item]
getChildNodes	Return array of node's children

Table D–3 XML Parser for C: DOM API Functions (Cont.)

Function	Brief Description
getContentModel	Returns the content model for an element from the DTD [DOM extension]
getDocument	Return top-level DOCUMENT node [DOM extension]
getDocumentElemen	Return highest-level (root) ELEMENT node
getDocType	Returns current DTD
getDocTypeEntities	Returns array of DTD's general entities
getDocTypeName	Returns name of DTD
getDocTypeNotations	Returns array of DTD's notations
getElementsByTagName	Returns list of elements with matching name
getEntityNotation	Returns an entity's NDATA [DOM getNotation]
getEntityPubID	Returns an entity's public ID [DOM getPublicId]
getEntitySysID	Returns an entity's system ID [DOM getSystemId]
getFirstChild	Returns the first child of a node
getImplementation	Returns DOM-implementation structure (if defined)
getLastChild	Returns the last child of a node
getModifier	Returns a content model node's '?', '*', or '+' modifier [DOM extension]
getNextSibling	Returns a node's next sibling
getNamedItem	Returns the named node from a list of nodes
getNodeMapLength	Returns number of entries in a NodeMap [DOM getLength]
getNodeName	Returns a node's name
getNodeType	Returns a node's type code (enumeration)
getNodeValue	Returns a node's "value", its character data
getNotationPubID	Returns a notation's public ID [DOM getPublicId]
getNotationSysID	Returns a notation's system ID [DOM getSystemId]
getOwnerDocument	Returns the DOCUMENT node containing the given node

Table D–3 XML Parser for C: DOM API Functions (Cont.)

Function	Brief Description
getPIData	Returns a processing instruction's data [DOM getData]
getPITarget	Returns a processing instruction's target [DOM getTarget]
getParentNode	Returns a node's parent node
getPreviousSibling	Returns a node's "previous" sibling
getTagName	Returns a node's "tagname", same as name for now
hasAttributes	Determines if element node has attributes [DOM extension]
hasChildNodes	Determines if node has children
hasFeature	Determines if DOM implementation supports a specific feature
insertBefore	Inserts a new child node before the given reference node
insertData	Inserts new character data into a node's existing data
isStandalone	Determines if document is standalone [DOM extension]
nodeValid	Validates a node against the current DTD [DOM extension]
normalize	Normalize a node by merging adjacent TEXT nodes
numAttributes	Returns number of element node's attributes [DOM extension]
numChildNodes	Returns number of node's children [DOM extension]
removeAttribute	Removes an element's attribute given its names
removeAttributeNode	Removes an element's attribute given its pointer
removeChild	Removes a node from its parents list of children
removeNamedItem	Removes a node from a list of nodes given its name
replaceChild	Replaces one node with another
replaceData	Replaces a substring of a node's character data with another string setAttribute Sets (adds or replaces) a new attribute for an element node given the attribute's name and value setAttributeNode Sets (adds or replaces) a new attribute for an element node given a pointer to the new attribute
setNamedItem	Sets (adds or replaces) a new node in a parent's list of children
setNodeValue	Sets a node's "value" (character data)

Table D–3 XML Parser for C: DOM API Functions (Cont.)

Function	Brief Description
setPIData	Sets a processing instruction's data [DOM setData]

XML Parser for C: Namespace API Functions

Table D–4 lists the XML Parser for C, Namespace functions.

Table D–4 XML Parser for C: Namespace API Functions

Function	Brief Description
getAttrLocal(xmlattr *attrs)	Returns attribute local name
getAttrNamespace(xmlattr *attr)	Returns attribute namespace (URI)
getAttrPrefix(xmlattr *attr)	Returns attribute prefix
getAttrQualifiedName(xmlattr *attr)	Returns attribute fully qualified name
getNodeLocal(xmlnode *node)	Returns node local name
getNodeNamespace(xmlnode *node)	Returns node namespace (URI)
getNodePrefix(xmlnode *node)	Returns node prefix
getNodeQualifiedName(xmlnode *node)	Returns node qualified name

XML Parser for C: XSLT API Functions

Table D–5 lists the XML Parser for C, XSLT functions.

Table D–5 XML Parser for C: XSLT API Functions

Function	Brief Description
xslprocess() xslprocess(xmlctx *docctx, xmlctx *xslctx, xmlctx *resctx, xmlnode **result)	Processes XSL Stylesheet with XML document source and returns success or an error code.

XML Parser for C: SAX API Functions

Table D–6 lists the XML Parser for C, SAX API functions.

Table D–6 XML Parser for C: SAX API Functions

SAX Function	Brief Description
characters(void *ctx, const oratext *ch, size_t len)	Receive notification of character data inside an element.
endDocument(void *ctx)	Receive notification of the end of the document.
endElement(void *ctx, const oratext *name)	Receive notification of the end of an element.
ignorableWhitespace(void *ctx, const oratext *ch, size_t len)	Receive notification of ignorable whitespace in element content.
notationDecl(void *ctx, const oratext *name, const oratext *publicId, const oratext *systemId)	Receive notification of a notation declaration.
processingInstruction(void *ctx, const oratext *target, const oratext *data)	Receive notification of a processing instruction.
startDocument(void *ctx)	Receive notification of the beginning of the document.
startElement(void *ctx, const oratext *name, const struct xmlattrs *attrs)	Receive notification of the start of an element.
unparsedEntityDecl(void *ctx, const oratext *name, const oratext *publicId, const oratext *systemId, const oratext *notationName)	Receive notification of an unparsed entity declaration.
Non-SAX Callback Functions	
nsStartElement(void *ctx, const oratext *qname, const oratext *local, const oratext *namespace, const struct xmlattrs *attrs)	Receive notification of the start of a namespace for an element.

E

XDK for C++: Specifications and Cheat Sheet

This appendix contains the following sections:

XML Parser for C++ Specifications

Oracle provides a set of XML parsers for Java, C, C++, and PL/SQL. Each of these parsers is a stand-alone XML component that parses an XML document (or a standalone DTD) so that it can be processed by an application. Library and command-line versions are provided supporting the following standards and features:

- DOM (Document Object Model) support is provided compliant with the W3C DOM 1.0 Recommendation. These APIs permit applications to access and manipulate an XML document as a tree structure in memory. This interface is used by such applications as editors.
- SAX (Simple API for XML) support is also provided compliant with the SAX 1.0 specification. These APIs permit an application to process XML documents using an event-driven model.
- Support is also included for W3C recommendation for XML Namespaces 1.0 thereby avoiding name collisions, increasing reusability and easing application integration.
- Supports validation and non-validation modes
- Supports W3C XML 1.0 Recommendation
- Integrated support for W3C XSLT 1.0 Recommendation

Validating and Non-Validating Mode Support

The XML Parser for C++ can parse XML in validating or non-validating modes.

- In *non-validating mode*, the parser verifies that the XML is well-formed and parses the data into a tree of objects that can be manipulated by the DOM API.
- In *validating mode*, the parser verifies that the XML is well-formed and validates the XML data against the DTD (if any).

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

Example Code

See Chapter 22, "Using XML Parser for C++" for example code and suggestions on how to use the XML Parser for C++.

Online Documentation

Documentation for Oracle XML Parser for C++ is located in the $ORACLE_HOME/xdk/cpp/parser/doc directory.

Release Specific Notes

The readme.html file in the root directory of the archive contains release specific information including bug fixes, API additions, and so on.

The Oracle XML parser for C++ is written in C with C++ wrappers. It will check if an XML document is well-formed, and optionally validate it against a DTD. The parser will construct an object tree which can be accessed via a DOM interface or operate serially via a SAX interface.

Standards Conformance

XML Parser for C++ conforms to the following standards:

- The W3C recommendation for Extensible Markup Language (XML) 1.0 at http://www.w3.org/TR/1998/REC-xml-19980210
- The W3C recommendation for Document Object Model Level 1 1.0 at http://www.w3.org/TR/REC-DOM-Level-1/
- The W3C proposed recommendation for Namespaces in XML at http://www.w3.org/TR/1998/PR-xml-names-19981117
- The Simple API for XML (SAX) 1.0 at http://www.megginson.com/SAX/index.html
- The W3C Recommendation for XSL Transform 1.0 at http://www.w3.org/TR/xslt

Supported Character Set Encodings

XML Parser for C++ supports documents in the following encodings, in addition to the ones specified in Appendix A, "Character Sets", of *Oracle8i National Language Support Guide*:

- BIG 5
- EBCDIC-CP-*
- EUC-JP
- EUC-KR

- GB2312
- ISO-2022-JP
- ISO-2022-KR
- ISO-8859-1, ISO-8859-2, ISO-8859-3, ..., ISO-8859-9
- ISO-10646-UCS-2
- ISO-10646-UCS-4
- KOI8-R
- Shift_JIS
- US-ASCII
- UTF-8
- UTF-16

Default: The default encoding is UTF-8. It is recommended that you set the default encoding explicitly if using only single byte character sets (such as US-ASCII or any of the ISO-8859 character sets) for performance up to 25% faster than with multibyte character sets, such as UTF-8.

XML Parser for C++ Revision History

Table E–1 lists the XML Parser for C++ revision history.

Table E–1 XML Parser for C++: Revision History

Revision	Description
Oracle XML Parser 2.0.4.0.0 (C++)	First production v2 release. Changes are mainly bug fixes. For XML parser, the following bugs were fixed: ■ 1352943 XMLPARSE() SOMETIMES CHOKES ON FILENAMES ■ 1302311 PROBLEM WITH PARAMETER ENTITY PROCESSING ■ 1323674 INCONSISTENT ERROR HANDLING IN THE C XML PARSER ■ 1328871 LPXPRINTBUFFER UNCONDITIONALLY PREPENDS XML COMMENT TO OUTPUT ■ 1349962 USING FREED MEMORY LOCATION CAUSES TLPXVNSA31.DIF oraxmldom.h was renamed to oradom.h
	For the XSLT processor, the following bugs were fixed: ■ 1225546 USELESS ERROR MESSAGE NEEDS DETAIL ■ 1267616 TLPXST14.DIF: REPLACE DBL_MAX WITH SBIG_ORAMAXVAL IN LPXXP.C:LPXXPSUBSTRING() ■ 1289228 ERROR CONTEXT REQUIRED FOR DEBUGGING: FILE NAME, LINE#, FUNCTION, ETC
	■ 1289214 XSL:CHOOSE DOESN'T WORK ■ 1298028 XPATH CONSTRUCT NOT(POSITION()=LAST()) NOT WORKING ■ 1298193 XPATH FUNCTIONS DON'T PROVIDE IMPLICIT TYPE CONVERSION OF PARAMS ■ 1323665 C XML PARSER CANNOT SET BASE DIRECTORY OR URI FOR STYLESHEET PARSING ■ 1325452 SEVERE MEMORY CONSUMPTION / LEAK IN XSLPROCESS ■ 1333693 CHAINED TRANSFORMS WITH C XSL PROCESSOR DON'T WORK: LPX-00002

Table E–1 XML Parser for C++: Revision History (Cont.)

Revision	Description
Oracle XML Parser 2.0.3.0.0 (C++)	SAX memory usage: Smaller, and flat for any input size and multiple parses (memory leaks plugged). XSLT memory usage: Improved. Validation warnings: Validity Constraint (VC) errors have been changed to warnings and do not terminate parsing. For compatibilty with the old behavior (halt on warnings as well as errors), a new flag XML_FLAG_STOP_ON_ WARNING (or '-W' to the xml program) has been added. Performance improvements: Switch to finite automata VC structure validation yields 10% performance gain. HTTP support: HTTP URIs are now supported; look for FTP in the next release. For other access methods, the user may define their own callbacks with the new xmlaccess() API.
Oracle XML Parser 2.0.2.0.0 (C++)	XSLT improvements: Various bugs fixed in the XSLT processor; error messages are improved; xsl:number, xsl:sort, xsl:namespace-alias, xsl:decimal-format, forwards-compatible processing with xsl:version, and literal result element as stylesheet are now available; the following XSLT-specific additions to the core XPath library are now available: current(), format-number(), generate-id(), and system-property(). XML parser bug fixes: Some problems with validation and matching of start and end tags with SAX were fixed. Also, a bug with parameter entity processing in external entities was fixed.
Oracle XML Parser 2.0.1.0.0 (C++)	Performance improvements: Major performance improvement over the last, about two and a half times faster for UTF-8 parsing and about four times faster for ASCII parsing. Comparison timing against previous version for parsing (DOM) and validating various standalone files (SPARC Ultra 1 CPU time):

File size	Old UTF-8	New UTF-8	Speedup	Old ASCII	New ASCII	Speedup
42K	180ms	70ms	2.6	120ms	40ms	3.0
134K	510ms	210ms	2.4	450ms	100ms	4.5
247K	980ms	400ms	2.5	690ms	180ms	3.8
1M	2860ms	1130ms	2.5	1820ms	380ms	4.8
2.7M	10550ms	4100ms	2.6	7450ms	1930ms	3.9
10.5M	42250ms	16400ms	2.6	29900ms	7800ms	3.8

	Conformance improvements: Stricter conformance to the XML 1.0 spec yields higher scores on standard test suites (Jim Clark, Oasis, etc).

Table E–1 XML Parser for C++: Revision History (Cont.)

Revision	Description
	Lists, not arrays: Internal parser data structures are now uniformly lists; arrays have been dropped. Therefore, access is now better suited to a firstChild / nextSibling style loop instead of numChildNodes / getChildNode. DTD parsing:A new API call xmlparsedtd() is added which parses an external DTD directly, without needing an enclosing document. Used mainly by the Class Generator.
	Error reporting: Error messages are improved and more specific, with nearly twice as many as before. Error location is now described by a stack of line number / entity pairs, showing the final location of the error and intermediate inclusions (e.g. line X of file, line Y of entity). NOTE: You must use the new error message file (lpxus.msb) provided with this release; the error message file provided with earlier releases is incompatible. See below. XSL improvements: Various bugs fixed in the XSLT processor; xsl:call-template is now fully supported.
Oracle XML Parser 2.0.1.0.0 (C++)	Performance improvements: Major performance improvement over the last, about two and a half times faster for UTF-8 parsing and about four times faster for ASCII parsing. Comparison timing against previous version for parsing (DOM) and validating various standalone files (SPARC Ultra 1 CPU time):File sizeOld UTF-8New UTF-8SpeedupOld ASCIINew ASCIISpeedup42K180ms70ms2.6120ms40ms3.0134K510ms210ms2.4450ms100ms4.5247K980ms400ms2.5690ms180ms3.81M2860ms1130ms2.51820ms380ms4.82.7M10550ms4100ms2.67450ms1930ms3.910.5M42250ms16400ms2.629900ms7800ms3.8
	Conformance improvements: Stricter conformance to the XML 1.0 spec yields higher scores on standard test suites (Jim Clark, Oasis, etc). Lists, not arrays: Internal parser data structures are now uniformly lists; arrays have been dropped. Therefore, access is now better suited to a firstChild / nextSibling style loop instead of numChildNodes / item.
	DTD parsing:A new method XMLParser::xmlparseDTD() is added which parses an external DTD directly, without needing an enclosing document. Used mainly by the Class Generator.

Table E–1 XML Parser for C++: Revision History (Cont.)

Revision	Description
	Error reporting: Error messages are improved and more specific, with nearly twice as many as before. Error location is now described by a stack of line number/entity pairs, showing the final location of the error and intermediate inclusions (e.g. line X of file, line Y of entity).
	NOTE: Use the new error message file (lpxus.msb) provided with this release; the error message file provided with earlier releases is incompatible. See below. XSL improvements: Various bugs fixed in the XSLT processor; xsl:call-template is now fully supported.
Oracle XML Parser 2.0.0.0.0 (C++)	The Oracle XML v2 parser is a beta release and is written in C, with a C++ wrapper. The main difference from the Oracle XML v1 parser is the ability to format the XML document according to a stylesheet via an integrated an XSLT processor. The XML parser will check if an XML document is well-formed, and optionally validate it against a DTD. The parser will construct an object tree which can be accessed via a DOM interface or operate serially via a SAX interface.

XML Parser for C++: XMLParser() API

Table E–2 lists the main XML Parser for C++,class XMLParser() methods with a brief description of each. XMLParser() class contains top-level methods that do the following:

- Invoke the parser
- Return high-level information about a document

Table E–2 XML Parser for C++ : XMLParser() Class

XMLParser() Method	Description
xmlinit	Initialize XML parser uword xmlinit(oratext *encoding, void (*msghdlr)(void *msgctx, oratext *msg, ub4 errcode), void *msgctx, lpxsaxcb *saxcb, void *saxcbctx, oratext *lang)
xmlterm	Terminate XML parser
xmlparse	Parse a document from a file
xmlparseBuffer	Parse a document from a buffer
getContent	Returns the content model for an element
getModifier	Returns the modifier ('?', '*' or '+') for a content-model node
getDocument	Returns the root node of a parsed document
getDocumentElement	Returns the root element (node) of a parsed document
getDocType	eturns the document type string
isStandalone	Returns the value of the standalone flag. Returns TRUE if the document is specified as standalone on the <?xml?> line, FALSE otherwise.

XML Parser for C++: DOM API

Table E–3 lists the XML Parser for C ++ DOM API methods a brief description of each.

Table E–3 XML Parser for C++: DOM API Classes (SubClasses)

Class (Subclass)	Methods	Description
Attr (Node)		
This class contains methods for accessing the name and value of a single document node attribute.		
	getName	Return name of attribute
	getValue	Return "value" (definition) of attribute
	getSpecified	Return attribute's "specified" flag value
	setValue	Set an attribute's value
CDATASection (Text)		
This class implements the CDATA node type, a subclass of Text. There are no methods.		
CharacterData (Node)		
This class contains methods for accessing and modifying the data associated with text nodes.		
	appendData	Append a string to this node's data
	deleteData	Remove a substring from this node's data
	getData	Get data (value) of a text node
	getLength	Return length of a text node's data
	insertData	nsert a string into this node's data
	replaceData	Replace a substring in this node's data
	substringData	Fetch a substring of this node's data
Comment (CharacterData)		
This class implements the COMMENT node type, a subclass of CharacterData. There are no methods.		
Document (Node)		
This class contains methods for creating and retrieving nodes.		
	createAttribute	Create an ATTRIBUTE node
	createCDATASection	Create a CDATA node

Table E–3 XML Parser for C++: DOM API Classes (SubClasses) (Cont.)

Class (Subclass)	Methods	Description
	createComment	Create a COMMENT node
	createDocumentFragment	Create a DOCUMENT_FRAGMENT node
	createElement	Create an ELEMENT node
	createEntityReference	Create an ENTITY_REFERENCE node
	createProcessingInstruction	Create a PROCESSING_INSTRUCTION node
	createTextNode	Create a TEXT node
	getElementsByTagName	Select nodes based on tag name
	getImplementation	Return DTD for document
DocumentFragment (Node)		
This class implements the DOCUMENT_FRAGMENT node type, a subclass of Node.		
DocumentType (Node)		
This class contains methods for accessing information about the Document Type Definition (DTD) of a document.		
	getName	R eturn name of DTD
	getEntities	Return NamedNodeMap of DTD's (general) entities
	getNotations	Return NamedNodeMap of DTD's notations
DOMImplementation		
This class contains methods relating to the specific DOM implementation supported by the parser.		
	hasFeature	Detect if the named feature is supported
	Element (Node	This class contains methods pertaining to element nodes.
	getTagName	Return the node's tag name
	getAttribute	Select an attribute given its name
	setAttribute	Create a new attribute given its name and value
	removeAttribute	Remove an attribute given its name
	getAttributeNode	Remove an attribute given its name
	setAttributeNode	Add a new attribute node
	removeAttributeNode	Remove an attribute node

Table E–3 XML Parser for C++: DOM API Classes (SubClasses) (Cont.)

Class (Subclass)	Methods	Description
	getElementsByTagName	Return a list of element nodes with the given tag name
	normalize	"Normalize" an element (merge adjacent text nodes)
Entity (Node)		
This class implements the ENTITY node type, a subclass of Node.		
	getNotation	NameReturn entity's NDATA (notation name)
	getPublicId	Return entity's public ID
	getSystemId	Return entity's system ID
EntityReference (Node)		
This class implements the ENTITY_REFERENCE node type, a subclass of Node.		
NamedNodeMap		
This class contains methods for accessing the number of nodes in a node map and fetching individual nodes.		
	item	Return nth node in map
	getLength	Return number of nodes in map
	getNamedItem	Select a node by name
	setNamedItem	Set a node into the map
	getLength	Remove the named node from map
Node		
This class contains methods for details about a document node		
	appendChild	Append a new child to the end of the current node's list of children
	cloneNode	Clone an existing node and optionally all its children
	getAttributes	Return structure contains all defined node attributes
	getChildNode	Return specific indexed child of given node
	getChildNodes	Return structure contains all child nodes of given node
	getFirstChild	Return first child of given node
	getLastChild	Return last child of given node
	getLocal	Returns the local name of the node

Table E–3 XML Parser for C++: DOM API Classes (SubClasses) (Cont.)

Class (Subclass)	Methods	Description
	getNamespace	Return a node's namespace
	getNextSibling	Return a node's next sibling
	getName	Return name of node
	getType	Return numeric type-code of node
	getValue	Return "value" (data) of node
	getOwnerDocument	Return document node which contains a node
	getParentNode	Return parent node of given node
	getPrefix	Returns the namespace prefix for the node
	getPreviousSibling	Returns the previous sibling of the current node
	getQualifiedName	Return namespace qualified node of given node
	hasAttributes	Determine if node has any defined attributes
	hasChildNodes	Determine if node has children
	insertBefore	Insert new child node into a node's list of children
	numChildNodes	Return count of number of child nodes of given node
	removeChild	Remove a node from the current node's list of children
	replaceChild	Replace a child node with another
	setValue	Sets a node's value (data)
NodeList		
This class contains methods for extracting nodes from a NodeList		
	item	Return nth node in list
	getLength	Return number of nodes in list
Notation (Node)		
This class implements the NOTATION node type, a subclass of Node.		
	getData	Return notation's data
	getTarget	Return notation's target
	setData	Set notation's data

Table E–3 XML Parser for C++: DOM API Classes (SubClasses) (Cont.)

Class (Subclass)	Methods	Description
ProcessingInstruction (Node)		
This class implements the PROCESSING_INSTRUCTION node type, a subclass of Node.		
	getData	Return the PI's data
	getTarget	Return the PI's target
	setData	Set the PI's data
Text (CharacterData)		
This class contains methods for accessing and modifying the data associated with text nodes (subclasses CharacterData).		
	splitText	Get data (value) of a text node

XML Parser for C++: XSLT API

XSLT is a language for tranforming XML documents into other XML documents. It is designed for use as part of XSL, which is a stylesheet language for XML. In addition to XSLT, XSL includes an XML vocabulary for specifying formatting. XSL specifies the styling of an XML document by using XSLT to describe how the document is transformed into another XML document that uses the formatting vocabulary.

XSLT is also designed to be used independently of XSL. However, XSLT is not intended as a completely general-purpose XML transformation language. Rather it is designed primarily for the kinds of transformation that are needed when XSLT is used as part of XSL.

A transformation expressed in XSLT describes rules for transforming a source tree into a result tree. The transformation is achieved by associating patterns with templates. A pattern is matched against elements in the source tree. A template is instantiated to create part of the result tree. The result tree is separate from the source tree. The structure of the result tree can be completely different from the structure of the source tree. In constructing the result tree, elements from the source tree can be filtered and reordered, and arbitrary structure can be added.

Stylesheets

A transformation expressed in XSLT is called a stylesheet. This is because, in the case when XSLT is transforming into the XSL formatting vocabulary, the transformation functions as a stylesheet.

A stylesheet contains a set of template rules. A template rule has two parts:

- A pattern which is matched against nodes in the source tree
- A template which can be instantiated to form part of the result tree. his allows a stylesheet to be applicable to a wide class of documents that have similar source tree structures.

How StylesheetTemplates are Processed

A template is instantiated for a particular source element to create part of the result tree. A template can contain elements that specify literal result element structure. A template can also contain elements from the XSLT namespace that are instructions for creating result tree fragments. When a template is instantiated, each instruction is executed and replaced by the result tree fragment that it creates.

Instructions can select and process descendant source elements. Processing a descendant element creates a result tree fragment by finding the applicable template rule and instantiating its template. Note that elements are only processed when they have been selected by the execution of an instruction. The result tree is constructed by finding the template rule for the root node and instantiating its template.

A software module called an XSL processor reads XML documents and transforms them into other XML documents with different styles.

XML Parser for C++ implementation of the XSL processor follows the XSL Transformations standard (version 1.0, November 16, 1999) and includes the required behavior of an XSL processor as specified in the XSLT specification.

Table E–4 lists the XSLProcessor class methods and syntax summary.

Table E–4 XML Parser for C++: XSLProcessor Class

Class	Method
XSLProcessor This class contains top-level methods for invoking the XSL processor.	xslprocess() Processes an XSL stylesheet with an XML document source. Syntax: uword xslprocess(XMLParser *docctx, XMLParser *xslctx, XMLParser *resctx, Node **result); where: docctx (IN / OUT) -- The XML document context xslctx (IN) -- The XSL stylesheet context resctx (IN) -- The result document fragment context result (IN / OUT) -- The result document fragment node

XML Parser for C++: SAX API

The SAX API is based on callbacks. Instead of the entire document being parsed and turned into a data structure which may be referenced (by the DOM interface), the SAX interface is serial. As the document is processed, appropriate SAX user callback functions are invoked. Each callback function returns an error code, zero meaning success, any non-zero value meaning failure. If a non-zero code is returned, document processing is stopped.

To use SAX, an xmlsaxcb structure is initialized with function pointers and passed to the xmlinit() call. A pointer to a user-defined context structure may also be included; that context pointer will be passed to each SAX function.

This SAX functionality is identical to the XML Parser for C version.

Table E–5 lists the XML Parser for C++, SAX API functions.

Table E–5 XML Parser for C++: SAX API Functions

SAX Function	Brief Description
characters(void *ctx, const oratext *ch, size_t len)	Receive notification of character data inside an element.
endDocument(void *ctx)	Receive notification of the end of the document.
endElement(void *ctx, const oratext *name)	Receive notification of the end of an element.

Table E–5 XML Parser for C++: SAX API Functions

SAX Function	Brief Description
ignorableWhitespace(void *ctx, const oratext *ch, size_t len)	Receive notification of ignorable whitespace in element content.
notationDecl(void *ctx, const oratext *name, const oratext *publicId, const oratext *systemId)	Receive notification of a notation declaration.
processingInstruction(void *ctx, const oratext *target, const oratext *data)	Receive notification of a processing instruction.
startDocument(void *ctx)	Receive notification of the beginning of the document.
startElement(void *ctx, const oratext *name, const struct xmlattrs *attrs)	Receive notification of the start of an element.
unparsedEntityDecl(void *ctx, const oratext *name, const oratext *publicId, const oratext *systemId, const oratext *notationName)	Receive notification of an unparsed entity declaration.
Non-SAX Callback Functions	
nsStartElement(void *ctx, const oratext *qname, const oratext *local, const oratext *namespace, const struct xmlattrs *attrs)	Receive notification of the start of a namespace for an element.

XML C++ Class Generator Specifications

Working in conjunction with the XML Parser for C++, the XML Class Generator generates a set of C++ source files based on an input DTD. The generated C++ source files can then be used to construct, optionally validate, and print a XML document that is compliant to the DTD specified. The Class Generator supports validation mode to assist debugging.

Input to the XML C++ Class Generator

Input is an XML document containing a DTD. The document body itself is ignored; only the DTD is relevant, though the dummy document must conform to the DTD. The underlying XML parser only accepts file names for the document and associated external entities. In future releases, no dummy document will be required, and URIs for additional protocols will be accepted.

Character Set Support

The following lists supported Character Set Encoding for files input to XML C++ Class Generator. These are in addition to the character sets specified in Appendix A, "Character Sets", of *Oracle8i National Language Support Guide*.

- BIG 5
- EBCDIC-CP-*
- EUC-JP
- EUC-KR
- GB2312
- ISO-2022-JP
- ISO-2022-KR
- ISO-8859-1, ISO-8859-2, ISO-8859-3, ..., ISO-8859-9
- ISO-10646-UCS-2
- ISO-10646-UCS-4
- KOI8-R
- Shift_JIS
- US-ASCII
- UTF-8

- UTF-16

Default: The default encoding is UTF-8. It is recommended that you set the default encoding explicitly if using only single byte character sets (such as US-ASCII or any of the ISO-8859 character sets) for performance up to 25% faster than with multibyte character sets, such as UTF-8.

Output to XML C++ Class Generator

XML Parser for C++ output is a pair of C++ source files, .cpp and .h, named after the DTD. Constructors are provided for each class (element) that allow an object to be created in two different ways: initially empty, then adding the children or data after the initial creation, or created with the initial full set of children or initial data. A method is provided for #PCDATA (and Mixed) elements to set the data and, when appropriate, set an element's attributes.

Standards Conformance

XML C++ Class Generator conforms to the following "Standards":

- The W3C recommendation for Extensible Markup Language (XML) 1.0
- The W3C recommendation for Document Object Model Level 1 1.0
- The W3C proposed recommendation for Namespaces in XML
- The Simple API for XML (SAX) 1.0

Directory Structure

The XML C++ Class Generator has the following file and directory structure:

```
license.html licensing agreement
bin/         Standalone Class Generator "xmlcg"
doc/         API documentation
include/     Header files
lib/         XML and support libraries
mesg/        Error message files (including cause/action information in the
            .msg)
sample/      Example usage
```

Table E–6 lists the libraries included with XML C++ Class Generator.

Table E–6 XML C++ Class Generator LIbraries

XML C++ Class Generator Library	Description
libxml8.a	XML Parser / XSL Processor
libxmlg8.a	XML Class Generator
libxmlc8.a	Compatibility library needed to link with Oracle 8.1.5
libcore8.a	CORE functions
libnls8.a	National Language Support

F

XDK for PL/SQL: Specifications and Cheat Sheets

This Appendix describes Oracle XDK for PL/SQL specifications and includes syntax cheat sheets. It contains the following sections:

XML Parser for PL/SQL

XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constraints on the storage layout and logical structure.

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

Oracle XML Parser Features

The XML Parser for PL/SQL parses an XML document (or a standalone DTD) so that it can be processed by an application. Library and command-line versions are provided supporting the following standards and features:

- DOM (Document Object Model) support is provided compliant with the W3C DOM 1.0 Recommendation. These APIs permit applications to access and manipulate an XML document as a tree structure in memory. This interface is used by such applications as editors.
- SAX (Simple API for XML) support is also provided compliant with the SAX 1.0 specification. These APIs permit an application to process XML documents using an event-driven model.
- Support is also included for XML Namespaces 1.0 thereby avoiding name collisions, increasing reusability and easing application integration.
- Able to run on Oracle*8i* and Internet Application Server (*i*AS)
- 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 24, "Using XML Parser for PL/SQL" for example code and suggestions on how to use the XML Parsers.

IXML Parser for PL/SQL Directory Structure

The following lists the XML Parser for PL/SQL directory structure in $ORACLE_HOME/xdk/plsql/parser:

- Windows NT
  - license.html - copy of license agreement
  - readme.html - release and installation notes
  - doc\ - directory for parser apis.
  - lib\ - directory for parser sql and class files
  - sample\ - sample code
- UNIX
  - license.html — copy of license agreement
  - readme.html — release and installation notes
  - doc/ — directory for parser apis
  - lib/ — directory for parser sql and class files
  - sample/ — sample code files

DOM and SAX APIs

XML APIs generally fall into two categories: event-based and tree-based. An event-based API (such as SAX) uses callbacks to report parsing events to the application. The application deals with these events through customized event handlers. Events include the start and end of elements and characters. Unlike tree-based APIs, event-based APIs usually do not build in-memory tree representations of the XML documents. Therefore, in general, SAX is useful for applications that do not need to manipulate the XML tree, such as search operations, among others. For example, the following XML document:

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

Becomes a series of linear events:

```
start document
start element: 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 8i database may be used.

Requirements

Oracle*8i* 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 *Oracle8i XML Reference*.

Release Specific Notes

The Oracle XML parser for PL/SQL is an early adopter release and is written in PL/SQL and Java. It will check if an XML document is well-formed and, optionally, if it is valid. The parser will construct an object tree which can be accessed via PLSQL interfaces.

Standards Conformance

The parser conforms to the following standards:

W3C recommendation for Extensible Markup Language (XML) 1.0 at http://www.w3.org/TR/1998/REC-xml-19980210

W3C recommendation for Document Object Model Level 1 1.0 at http://www.w3.org/TR/REC-DOM-Level-1/

The parser currently does not currently have SAX or Namespace support. These will be made available in a future version.

Error Recovery

The parser also provides error recovery. It will recover from most errors and continue processing until a fatal error is encountered.

Important note: The contents of both the Windows and UNIX versions are identical. They are simply archived differently for operating system compatibility and your convenience.

XML Parser for PL/SQL: Parser() API

Table F–1 lists the XML Parser for PL/SQL Parser() API functions.

Table F–1 XML Parser for PL/SQL: Parser() API

Parser() Functions	Description
parse(VARCHAR2)	Parses xml stored in the given url/file and returns the built DOM Document
newParser	Returns a new parser instance
parse(Parser, VARCHAR2)	Parses xml stored in the given url/file
parseBuffer(Parser, VARCHAR2)	Parses xml stored in the given buffer
parseClob(Parser, CLOB)	Parses xml stored in the given clob
parseDTD(Parser, VARCHAR2, VARCHAR2)	Parses xml stored in the given url/file
parseDTDBuffer(Parser, VARCHAR2, VARCHAR2)	Parses xml stored in the given buffer
parseDTDClob(Parser, CLOB, VARCHAR2)	Parses xml stored in the given clob
setBaseDir(Parser, VARCHAR2)	Sets base directory used to resolve relative urls
showWarnings(Parser, BOOLEAN)	Turn warnings on or off

Table F–1 XML Parser for PL/SQL: Parser() API

Parser() Functions	Description
setErrorLog(Parser, VARCHAR2)	Sets errors to be sent to the specified file
setPreserveWhitespace(Parser, BOOLEAN)	Sets white space preserve mode
setValidationMode(Parser, BOOLEAN)	Sets validation mode
getValidationMode(Parser)	Gets validation mode
setDoctype(Parser, DOMDocumentType)	Sets DTD
getDoctype(Parser)	Gets DTD
getDocument(Parser)	Gets DOM document
freeParser(Parser)	Frees a Parser object

XML Parser for PL/SQL: XSL-T Processor API

Table F–2 lists the XML Parser for PL / SQL XSL-T Processor API functions.

for the following interfaces:

- Processor interface type: Processor()
- Stylesheet interface type: Stylesheet()

Table F–2 XML Parser for PL/SQL: XSL-T Processor() API Functions

XSL-T Processor Functions	Description
newProcessor	Returns a new processor instance
processXSL(Processor, Stylesheet, DOMDocument)	Transforms input XML document using given DOMDocument and stylesheet
processXSL(Processor, Stylesheet, DOMDocumentFragment)	Transforms input XML document fragment using given DOMDocumentFragment and stylesheet
showWarnings(Processor, BOOLEAN)	Turn warnings on or off
setErrorLog(Processor, VARCHAR2)	Sets errors to be sent to the specified file
newStylesheet(DOMDocument, VARCHAR2)	Returns a new stylesheet using the given DOMDocument and reference URL
newStylesheet(VARCHAR2, VARCHAR2)	Returns a new stylesheet using the given input and reference URLs
transformNode(DOMNode, Stylesheet)	Transforms a node in a DOM tree using the given stylesheet
selectNodes(DOMNode, VARCHAR2)	Selects nodes from a DOM tree which match the given pattern
selectSingleNodes(DOMNode, VARCHAR2)	Selects the first node from the tree that matches the given pattern
valueOf(DOMNode, VARCHAR2)	Retrieves the value of the first node from the tree that matches the given pattern
setParam(Stylesheet, VARCHAR2, VARCHAR2)	Sets the value of a top-level stylesheet parameter
removeParam(Stylesheet, VARCHAR2)	Remove a top-level stylesheet parameter
resetParams(Stylesheet)	Resets the top-level stylesheet parameters
freeStylesheet(Stylesheet)	Free a stylesheet object
freeProcessor(Processor)	Free a processor object

XML Parser for PL/SQL: W3C DOM API — Types

The Document Object Model (DOM) is an application programming interface (API) for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated. In the DOM specification, the term "document" is used in the broad sense - increasingly, XML is being used as a way of representing many different kinds of information that may be stored in diverse systems, and much of this would traditionally be seen as data rather than as documents. Nevertheless, XML presents this data as documents, and the DOM may be used to manage this data.

The XML Parser for PL/SQL W3C DOM APIs are listed on OTN at the following site:

http://technet.oracle.com/tech/xml/parser_plsql/index.htm

Table F–3 XML Parser for PL/SQL: W3C DOM API Types

Types	DOMException types	DOM interface types
DOM Node types	INDEX_SIZE_ERR	DOMNode
ELEMENT_NODE	DOMSTRING_SIZE_ERR	DOMNamedNodeMap
ATTRIBUTE_NODE	HIERARCHY_REQUEST_ERR	DOMNodeList
TEXT_NODE	WRONG_DOCUMENT_ERR	DOMAttr
CDATA_SECTION_NODE	INVALID_CHARACTER_ERR	DOMCDataSection
ENTITY_REFERENCE_NODE	NO_DATA_ALLOWED_ERR	DOMCharacterData
ENTITY_NODE	NO_MODIFICATION_ALLOWED_ERR	DOMComment
PROCESSING_INSTRUCTION_NODE	NOT_FOUND_ERR	DOMDocumentFragment
COMMENT_NODE	NOT_SUPPORTED_ERR	DOMElement
DOCUMENT_NODE	INUSE_ATTRIBUTE_ERR	DOMEntity
DOCUMENT_TYPE_NODE	DOMException types	DOMEntityReference
DOCUMENT_FRAGMENT_NODE	INDEX_SIZE_ERR	DOMNotation
NOTATION_NODE	DOMSTRING_SIZE_ERR	DOMProcessingInstruction
		DOMText

Table F–3 XML Parser for PL/SQL: W3C DOM API Types(Cont.)

Types	DOMException types	DOM interface types
		DOMImplementation
		DOMDocumentType
		DOMDocument

XML Parser for PL/SQL: W3C DOM API — Node Methods, Node Types, and DOM Interface Types

Node Methods

The following lists the DOM API Node methods:

- FUNCTION isNull(n DOMNode) RETURN BOOLEAN;
- FUNCTION makeAttr(n DOMNode) RETURN DOMAttr;
- FUNCTION makeCDataSection(n DOMNode) RETURN DOMCDataSection;
- FUNCTION makeCharacterData(n DOMNode) RETURN DOMCharacterData;
- FUNCTION makeComment(n DOMNode) RETURN DOMComment;
- FUNCTION makeDocumentFragment(n DOMNode) RETURN DOMDocumentFragment;
- FUNCTION makeDocumentType(n DOMNode) RETURN DOMDocumentType;
- FUNCTION makeElement(n DOMNode) RETURN DOMElement;
- FUNCTION makeEntity(n DOMNode) RETURN DOMEntity;
- FUNCTION makeEntityReference(n DOMNode) RETURN DOMEntityReference;
- FUNCTION makeNotation(n DOMNode) RETURN DOMNotation;
- FUNCTION makeProcessingInstruction(n DOMNode) RETURN DOMProcessingInstruction;
- FUNCTION makeText(n DOMNode) RETURN DOMText;
- FUNCTION makeDocument(n DOMNode) RETURN DOMDocument;

- PROCEDURE writeToFile(n DOMNode, fileName VARCHAR2);
- PROCEDURE writeToBuffer(n DOMNode, buffer IN OUT VARCHAR2);
- PROCEDURE writeToClob(n DOMNode, cl IN OUT CLOB);
- PROCEDURE writeToFile(n DOMNode, fileName VARCHAR2, charset VARCHAR2);
- PROCEDURE writeToBuffer(n DOMNode, buffer IN OUT VARCHAR2, charset VARCHAR2);
- PROCEDURE writeToClob(n DOMNode, cl IN OUT CLOB, charset VARCHAR2);
- FUNCTION getNodeName(n DOMNode) RETURN VARCHAR2;
- FUNCTION getNodeValue(n DOMNode) RETURN VARCHAR2;
- PROCEDURE setNodeValue(n DOMNode, nodeValue IN VARCHAR2);
- FUNCTION getNodeType(n DOMNode) RETURN NUMBER;
- FUNCTION getParentNode(n DOMNode) RETURN DOMNode;
- FUNCTION getChildNodes(n DOMNode) RETURN DOMNodeList;
- FUNCTION getFirstChild(n DOMNode) RETURN DOMNode;
- FUNCTION getLastChild(n DOMNode) RETURN DOMNode;
- FUNCTION getPreviousSibling(n DOMNode) RETURN DOMNode;
- FUNCTION getNextSibling(n DOMNode) RETURN DOMNode;
- FUNCTION getAttributes(n DOMNode) RETURN DOMNamedNodeMap;
- FUNCTION getOwnerDocument(n DOMNode) RETURN DOMDocument;
- FUNCTION insertBefore(n DOMNode, newChild IN DOMNode, refChild IN DOMNode) RETURN DOMNode;
- FUNCTION replaceChild(n DOMNode, newChild IN DOMNode, oldChild IN DOMNode)> RETURN DOMNode;
- FUNCTION removeChild(n DOMNode, oldChild IN DOMNode) RETURN DOMNode;
- FUNCTION appendChild(n DOMNode, newChild IN DOMNode) RETURN DOMNode;
- FUNCTION hasChildNodes(n DOMNode) RETURN BOOLEAN;

- FUNCTION cloneNode(n DOMNode, deep boolean) RETURN DOMNode;

DOM Node Types

The following lists the DOM API Node types:

- DOM Node types
- ELEMENT_NODE
- ATTRIBUTE_NODE
- TEXT_NODE
- CDATA_SECTION_NODE
- ENTITY_REFERENCE_NODE
- ENTITY_NODE
- PROCESSING_INSTRUCTION_NODE
- COMMENT_NODE
- DOCUMENT_NODE
- DOCUMENT_TYPE_NODE
- DOCUMENT_FRAGMENT_NODE
- NOTATION_NODE

DOMException Types

The following lists the DOMException types:

- INDEX_SIZE_ERR
- DOMSTRING_SIZE_ERR
- HIERARCHY_REQUEST_ERR
- WRONG_DOCUMENT_ERR
- INVALID_CHARACTER_ERR
- NO_DATA_ALLOWED_ERR
- NO_MODIFICATION_ALLOWED_ERR
- NOT_FOUND_ERR

- NOT_SUPPORTED_ERR
- INUSE_ATTRIBUTE_ERR

DOM Interface Types

The following lists the DOM Interface types:

- DOM interface types
- DOMNode
- DOMNamedNodeMap
- DOMNodeList
- DOMAttr
- DOMCDataSection
- DOMCharacterData
- DOMComment
- DOMDocumentFragment
- DOMElement
- DOMEntity
- DOMEntityReference
- DOMNotation
- DOMProcessingInstruction
- DOMText
- DOMImplementation
- DOMDocumentType
- DOMDocument

G

XML-SQL Utility (XSU) Specifications and Cheat Sheets

This appendix contains the following sections:

Installing XML-SQL Utility

Contents of the XSU Distribution

Table G–1 lists the XSU distribution archive (zip file) contents.

Table G–1 XSU Archive Contents

File (with relative location)	Description
relnotes.html	The release notes
env.csh	This files is a helper csh shell script which can set up all the environmental variables needed to run the utility correctly. The user must setup the directory information correctly (for example, point to the installed area for the JDK etc.)
env.bat	This file is the same as the env.csh except that it is written for the Windows platform.
lib/oraclexmlsql.jar	The jar file containing all the Java functions for the utility.
lib/xmlparserv2.jar	The Oracle XML parser V2 packaged with the utility.
lib/oraclexmlsqlload.csh (Unix) lib/oraclexmlsqlload.bat (Windows)	A csh and bat script to help load the utility into an Oracle database. These scripts call loadjava to load the jar file into the database and then run the xmlgenpkg.sql to create the PL/SQL front-end wrappers.
lib/xmlgenpkg.sql	This file contains the sql script for creating the PL/SQL front-end wrappers.

Installing XML-SQL Utility: Procedure

To install XML-SQL Utility (XSU) follow these steps:

1. Requirements. Check that you have the correct software requirements loaded.
2. Extract the XSU files
3. Set Up Your Environment Correctly: Client Side
   - CLASSPATH Settings
   - Ensure the database is up
4. Set Up Your Environment Correctly: Server Side

Requirements for Running XML-SQL Utility

There are two versions of the utility - XSU111.zip and XSU12.zip, one compatible for JDK 1.1.x and the other with JDK1.2 respectively.

XSU111 - XML-SQL Utility (for JDK 1.1.x)

This is the JDK 1.1.x compatible version. It needs Java JDK 1.1.x version and the JDBC 1.x compliant drivers. Oracle ships classes111.zip as the JDBC 1.x compliant driver. The utility is also dependent on the Oracle XML Parser V2 for parsing documents and for creating DOM nodes. To load this into the database, the database must be running Oracle8*i* JVM (JServer) version 8.1.5 or above.

XSU12 - XML-SQL Utility (for JDK 2.x)

This version of the utility is JDK1.2.x/JDK 2.x compliant. It needs Java JDK 1.2.x (or JDK2.x) to be available. The JDBC drivers need to be JDK1.2 compliant as well. Oracle ships classes12.zip as the JDBC2.0 compliant driver. The XSU utility is also dependent on the Oracle XML parser V2. The database version must be 8.1.6 or above, with Oracle8*i* JVM (JServer) enabled, for the utility to be uploaded in to the server and run inside the database.

XSU Requirements

Before installing the utility make sure that you choose the right version of the utility depending on your particular needs. For example, if you can only use the JDK1.1.x version, then download the XSU111.zip file. Ensure that you have the JDK and the JDBC drivers correctly downloaded and installed, if not already available.

Extract the XSU Files

After downloading the zip file, simply extract the contents to a directory of your choice, say C:\xml. The files will get expanded in to a subdirectory called *XSU111* or *XSU12* depending on the version of the utility.

Setting Up the Correct XSU Environment: Client Side

Depending on the way you plan to use the utility, on the client or server side, perform different set up tasks.

CLASSPATH Settings

If you are running XSU from the client side, you need to correct CLASSPATH settings. If these variables are already set correctly in your environment, then you need to only ensure that **oraclexmlsql.jar** is included in this CLASSPATH. You can do this by either editing your global settings or changing the settings for your current shell.

- *Global Settings*: To edit your global settings proceed as follows:
  - UNIX: Set the CLASSPATH variable in your *login* or *.tcsh* scripts. Simply edit them and include the path to the oraclexmlsql.jar file. For example,

    ```
    setenv CLASSPATH
    /private/john/xmlwork/XSU12/lib/oraclexmlsql.jar:$CLASSPATH
    ```

  - Windows: Modify the enviroment variable in the *System Properties* folder under the Settings / Control Panel in Windows to include the path. For example, if you expanded the utility under D:\xml, include D:\xml\XSU12\lib\oraclexmlsql.jar in the variable setting.
- *Current Shell Settings*: To modify the **settings for the particular shell** you are running in follow these steps:
  - UNIX: Edit the *env.csh* file Make sure that the path names in env.csh are correct and then *source* the *env.csh* file. This sets the environment variables correctly. If you are using a shell other than csh or tcsh, edit the file to use your shell's syntax.
  - Windows: Edit the *env.bat* file, and ensure that the variable settings are correct and then *execute* the file to set up the variables in the current environment.

  The latter operation only affects the current shell you are running in and consequently you have to *source* or *execute* these batch files for every different shell.

Database

Ensure that Your database is up and running.

After this, you are ready to use the utility from the client side.

Setting Up the Correct XSU Environment: Server Side

To use XSU's PL / SQL API, or write Java stored procedures on top of XSU Java APIs, perform the following steps:

1. Choose the database schema into which you want to upload XSU. The default schema is "scott". To change this schema, edit the **oraclexmlsql.xxx** file and modify the *USER_PASSWORD* macro file to the desired schema.
   - UNIX: Modify oraclexmlsqlload.csh file
   - Windows: Modify oraclexmlsqlload.bat file
2. Confirm that the Oracle database into which you are planning to load XSU is Java enabled and is up and running.
3. Execute the appropriate **oraclexmlsqlload.xxx** file: This does the following:
   - Loads Oracle XML Parser for Java, V2, into the database. If the Parser is already loaded into the database, you can comment out the line in oraclexmlsqlload.xxx which loads the Parser.
   - Loads **oraclexmlsql.jar**, the Java classes that make up XSU, into your Oracle database.
   - Executes the **xmlgenpkg.sql** SQL script. This creates the PL/SQL API on top of the loaded Java classes.
   - Executes **oraclexmltest.sql** which tests that the PL/SQL API is loaded and operational. As you run this, you will see SQLPlus come up and the various queries run and XML results displayed. If the tests fail to run, you need to re-load XSU after taking appropriate steps to fix the problems.

Now, you are ready to use the Java and PL/SQL APIs inside the Oracle server.

XML-SQL Utility (XSU) for Java Cheat Sheets

Table G–2, Table G–3, and Table G–4 list the XML-SQL Utility (XSU) for Java cheat sheets for the top level classes.

Table G–2 OracleXMLSave()

Member Summary	
Fields	
DATE_FORMAT	date format for use in setDateFormat(String)
DEFAULT_BATCH_SIZE	default insert batch size is 17
Constructors	
OracleXMLSave(Connection, String)	The public constructor for OracleXMLSave utility
Methods	
close()	It closes / deallocates all the context associated with this object.
deleteXML(Document)	Deletes records from the db. object corresponding to the records in an XML doc.
deleteXML(InputStream)	Deletes records from the db. object corresponding to the records in an XML doc.
Deletes records from the db. object corresponding to the records in an XML doc.	Deletes records from the db. object corresponding to the records in an XML doc.
deleteXML(URL)	Deletes records from the db. object corresponding to the records in an XML doc.
insertXML(Document)	Inserts an XML document into a specified table. Returns the number of rows processed.
insertXML(InputStream)	Inserts an XML document into a specified table. Returns the number of rows processed.
insertXML(String)	Inserts an XML document into a specified table. Returns the number of rows processed.
insertXML(URL)	Inserts an XML document into a specified table. Returns the number of rows processed.
setBatchSize(int)	This call changes the batch size used during DML operations.
setCommitBatch(int)	Sets the commit batch size.

Table G–2 OracleXMLSave() (Cont.)

Member Summary	
setDateFormat(String)	Sets the format of the date tags.
setIgnoreCase(boolean)	Used to tell XSU to ignores the case of the tag names when matching them with the column names of the table / view.
setKeyColumnList(String[])	Sets the list of columns to be used for identifying a particular row in the database table during update or delete.
setRowTag(String)	Names the tag used in the XML doc., to enclose the XML elements corresponding to each row value..
setUpdateColumnList(String[])	Set the column values to be updated.
updateXML(Document)	Updates the table given the XML document.
updateXML(InputStream)	Updates the table given the XML document.
updateXML(String)	Updates the table given the XML document.
updateXML(URL)	Updates the table given the XML document.

Table G–3 OracleXMLQuery()

Member Summary	Description
Fields	
DTD	The DTD is used to specified that the DTD is to be generated
ERROR_TAG	The ERROR_TAG specifies the default tag name for the ERROR document
MAXROWS_ALL	The MAXROWS_NONE specifies that all rows be included in the result
NONE	The NONE is used to specified that no DTD is to be generated
ROW_TAG	The ROW_TAG specifies the default tag name for the ROW elements
ROWIDATTR_TAG	The ROWIDATTR_TAG specifies the default tag name for the ROW elements
ROWSET_TAG	The ROWSET_TAG specifies the default tag name for the document
SKIPROWS_ALL	The SKIPROWS_ALL specifies that all rows be skipped in the result.
Constructors	
OracleXMLQuery(Connection, ResultSet)	Constructor for the OracleXMLQueryObject.

Table G–3 OracleXMLQuery() (Cont.)

Member Summary	Description
OracleXMLQuery(Connection, String)	Constructor for the OracleXMLQueryObject.
OracleXMLQuery(OracleXMLDataSet)	Constructor for the OracleXMLQueryObject.
Methods	
close()	Close any open resource.
getXMLDOM()	getXMLDOM returns a DOM representation of the XML document using the OracleXMLDocGenDOM class to generate the DOM tree
getXMLDOM(Node)	The root node is also passed in.
getXMLDOM(Node, int)	This method returns a Document object containing the XML doc.
getXMLMetaData(int, boolean)	This functions returns the DTD for the XML document which would have been generated by a getXML call.
getXMLString()	getXMLString returns a string representation of the XML document using the OracleXMLDocGenString class to create the string; generates no DTD.
getXMLString(int)	getXMLString returns a string representation of the XML document using the OracleXMLDocGenString class to create the string.
getXMLString(Node)	All the children created are appended to this root node.
getXMLString(Node, int)	This method returns a String object containing the XML doc.
keepObjectOpen(boolean)	The default behavior for all the getXML functions which DO NOT TAKE in a ResultSet object is to close the ResultSet object and Statement objects at the end of the call; this method allows for change of this behaviour. This is very useful when retreiving few records at a time because it preserves all the metadata.
setCollIdAttrName(String)	
setDataHeader(Reader, String)	Sets the XML data header.
setDateFormat(String)	Sets the format of the generated dates in the XML doc.
setEncoding(String)	Sets the encoding PI (processing instruction) in the XML doc.
setErrorTag(String)	setErrorTag sets the name of the error tag (default "ERROR") in case an error is raised.
setException(Exception)	Allows the user to pass in an exception, and have the XSU handle it.
setMaxRows(int)	This sets the maximum number of rows to be retreived from the query result after "skipRows" number of rows are skipped You can use one of MAXROWS_NONE, MAXROWS_ALL, MAXROWS_DEFAULT

Table G–3 OracleXMLQuery() (Cont.)

Member Summary	Description
setMetaHeader(Reader)	Sets the XML meta header.
setRaiseException(boolean)	setRaiseException function when called with the value true, will raise exceptions such as no-data found, instead of returning a NULL row out.
setRaiseNoRowsException(boolean)	Tells the XSU to throw or not to throw an OracleXMLNoRowsException in the case when for one reason or another, the XML doc generated is empty.
setRowIdAttrName(String)	Sets the name of the id attribute of the row enclosing tag.
setRowIdAttrValue(String)	Specifies the scalar column whose value is to be assigned to the id attribute of the row enclosing tag.
setRowsetTag(String)	Sets the tag to be used to enclose the xml dataset.
setRowTag(String)	Sets the tag to be used to enclose the xml element corresponding to a db.
setSkipRows(int)	Sets the number of rows to skip.
setStylesheetHeader(String)	Sets the stylesheet header (i.e. stylesheet processing instructions) in the generated XML doc.
setStylesheetHeader(String, String)	Sets the stylesheet header (i.e. stylesheet processing instructions) in the generated XML doc.
setXSLT(Reader, String)	Sets the stylesheet to be applied during XML generation.
setXSLT(String, String)	Sets the stylesheet to be applied during XML generation.
useLowerCaseTagNames()	This will set the case to be lower for all tag names.
useNullAttributeIndicator(boolean)	Specified weather to use an XML attribute to indicate NULLness; or to do it by omitting the inclusion of the particular entity in the XML document.
useUpperCaseTagNames()	This will set the case to be upper for all tag names.

Table G–4 OracleXMLSQLException()

Member Summary	Description
Constructors	
OracleXMLSQLException(Exception)	
OracleXMLSQLException(Exception, String)	

Table G–4 OracleXMLSQLException()(Cont.)

Member Summary	Description
OracleXMLSQLException(String)	
OracleXMLSQLException(String, Exception)	
OracleXMLSQLException(String, Exception, String)	
OracleXMLSQLException(String, int)	
OracleXMLSQLException(String, int, String)	
OracleXMLSQLException(String, String)	
Methods	
getErrorCode()	
getParentException()	returns the original exception, if there was one; otherwise, it returns null
getXMLErrorString()	prints the XML error string with the given error tag name
getXMLSQLErrorString()	prints the SQL parameters as well in the error message
setErrorTag(String)	Sets the error tag name which is then used by getXMLErrorString and getXMLSQLErrorString, to generate xml error reports

XML-SQL Utility (XSU) for PL/SQL Cheat Sheets

XML-SQL Utility (XSU) for PL/SQL offers the following PL/SQL packages:

- DBMS_XMLQuery -- provides DB_to_XML type functionality.
- DBMS_XMLSave -- provides XML_to_DB type functionality.

See Also: *Oracle8i XML Reference*

Glossary

API

Application Program Interface. See application program, definition interface.

application program interface (API)

A set of public programmatic interfaces that consist of a language and message format to communicate with an operating system or other programmatic environment, such as databases, Web servers, JVMs, and so forth. These messages typically call functions and methods available for application development.

application server

A server designed to host applications and their environments, permitting server applications to run. A typical example is OAS, which is able to host Java, C, C++, and PL/SQL applications in cases where a remote client controls the interface. See also Oracle Application Server.

attribute

A property of an element that consists of a name and a value separated by an equals sign and contained within the start tags after the element name. In this example, <Price units='USD'>5</Price>, units is the attribute and USD is its value, which must be in single or double quotes. Attributes may reside in the document or DTD. Elements may have many attributes but their retrieval order is not defined.

BC4J

Business Components for Java.

Business-to-Business (B2B)

A term describing the communication between businesses in the selling of goods and services to each other. The software infrastructure to enable this is referred to as an exchange.

Business-to-Consumer (B2C)

A term describing the communication between businesses and consumers in the selling of goods and services.

BFILES

External binary files that exist outside the database tablespaces residing in the operating system. BFILES are referenced from the database semantics, and are also known as External LOBs.

Binary Large Object (BLOB)

A Large Object datatype whose content consists of binary data. Additionally, this data is considered raw as its structure is not recognized by the database.

BLOB

See Binary Large Object.

callback

A programmatic technique in which one process starts another and then continues. The second process then calls the first as a result of an action, value, or other event. This technique is used in most programs that have a user interface to allow continuous interaction.

cartridge

A stored program in Java or PL/SQL that adds the necessary functionality for the database to understand and manipulate a new datatype. Cartridges interface through the Extensibility Framework within Oracle 8 or 8i. interMedia Text is just such a cartridge, adding support for reading, writing, and searching text documents stored within the database.

CDATA

See character data.

CDF

Channel Definition Format. Provides a way to exchange information about channels on the internet.

CGI

See Common Gateway Interface.

CSS

Cascading Style Sheets.

character data (CDATA)

Text in a document that should not be parsed is put within a CDATA section. This allows for the inclusion of characters that would otherwise have special functions, such as &, <, >, etc. CDATA sections can be used in the content of an element or in attributes.

Common Gateway Interface (CGI)

The generic acronym for the programming interfaces enabling Web servers to execute other programs and pass their output to HTML pages, graphics, audio, and video sent to browsers.

child element

An element that is wholly contained within another, which is referred to as its parent element. For example <Parent><Child></Child></Parent> illustrates a child element nested within its parent element.

Class Generator

A utility that accepts an input file and creates a set of output classes that have corresponding functionality. In the case of the XML Class Generator, the input file is a DTD and the output is a series of classes that can be used to create XML documents conforming with the DTD.

CLASSPATH

The operating system environmental variable that the JVM uses to find the classes it needs to run applications.

client-server

The term used to describe the application architecture where the actual application runs on the client but accesses data or other external processes on a server across a network.

Character Large Object (CLOB)

The LOB datatype whose value is composed of character data corresponding to the database character set. A CLOB may be indexed and searched by the interMedia Text search engine.

CLOB

See Character Large Object.

command line

The interface method in which the user enters in commands at the command interpreter's prompt.

Common Object Request Broker API (CORBA)

An Object Management Group standard for communicating between distributed objects across a network. These self-contained software modules can be used by applications running on different platforms or operating systems. CORBA objects and their data formats and functions are defined in the Interface Definition Language (IDL), which can be compiled in a variety of languages including Java, C, C++, Smalltalk and COBOL.

Common Oracle Runtime Environment (CORE)

The library of functions written in C that provides developers the ability to create code that can be easily ported to virtually any platform and operating system.

CORBA

See Common Object Request Broker.

Database Access Descriptor (DAD)

A DAD is a named set of configuration values used for database access. A DAD specifies information such as the database name or the SQL*Net V2 service name, the ORACLE_HOME directory, and NLS configuration information such as language, sort type, and date language.

datagram

A text fragment, which may be in XML format, that is returned to the requester embedded in an HTML page from a SQL query processed by the XSQL Servlet.

DOCTYPE

The term used as the tag name designating the DTD or its reference within an XML document. For example, <!DOCTYPE person SYSTEM "person.dtd"> declares the

root element name as person and an external DTD as person.dtd in the file system. Internal DTDs are declared within the DOCTYPE declaration.

Document Object Model (DOM)

An in-memory tree-based object representation of an XML document that enables programmatic access to its elements and attributes. The DOM object and its interface is a W3C recommendation. It specifies the Document Object Model of an XML Document including the APIs for programmatic access. DOM views the parsed document as a tree of objects.

Document Type Definition (DTD)

A set of rules that define the allowable structure of an XML document. DTDs are text files that derive their format from SGML and can either be included in an XML document by using the DOCTYPE element or by using an external file through a DOCTYPE reference.

DOM

See Document Object Model.

DTD

See Document Type Definition.

EDI

Electronic Data Interchange.

Enterprise Java Bean (EJB)

An independent program module that runs within a JVM on the server. CORBA provides the infrastructure for EJBs, and a container layer provides security, transaction support, and other common functions on any supported server.

element

The basic logical unit of an XML document that may serve as a container for other elements as children, data, attributes, and their values. Elements are identified by start-tags, <name> and end-tags</name> or in the case of empty elements, <name/>.

empty element

An element without text content or child elements. It may only contain attributes and their values. Empty elements are of the form <name/> or <name></name> where there is no space between the tags.

entity

A string of characters that may represent either another string of characters or special characters that are not part of the document's character set. Entities and the text that is substituted for them by the parser are declared in the DTD.

eXtensible Markup Language (XML)

An open standard for describing data developed by the W3C using a subset of the SGML syntax and designed for Internet use. Version 1.0 is the current standard, having been published as a W3C Recommendation in February 1998.

eXtensible Stylesheet Language (XSL)

The language used within stylesheets to transform or render XML documents. There are two W3C recommendations covering XSL stylesheets—XSL Transformations (XSLT) and XSL Formatting Objects (XSLFO).

XSL

(W3C) eXtensible Stylesheet Language, XSL consists of two W3C recommendations - XSL Transformations for transforming one XML document into another and XSL Formatting Objects for specifying the presentation of an XML document. XSL is a language for expressing stylesheets. It consists of two parts:

- A language for transforming XML documents (XSLT), and
- An XML vocabulary for specifying formatting semantics (XSL:FO).

An XSL stylesheet specifies the presentation of a class of XML documents by describing how an instance of the class is transformed into an XML document that uses the formatting vocabulary.

eXtensible Stylesheet Language Formatting Object (XSLFO)

The W3C standard specification that defines an XML vocabulary for specifying formatting semantics.

eXtensible Stylesheet Language Transformation (XSLT)

Also written as XSL-T. The XSL W3C standard specification that defines a transformation language to convert one XML document into another.

HTML

See Hypertext Markup Language.

HTTP

See Hypertext Transport Protocol.

hypertext

The method of creating and publishing text documents in which users can navigate between other documents or graphics by selecting words or phrases designated as hyperlinks.

Hypertext Markup Language (HTML)

The markup language used to create the files sent to Web browsers and that serves as the basis of the World Wide Web. The next version of HTML will be called xHTML and will be an XML application.

Hypertext Transport Protocol (HTTP)

The protocol used for transporting HTML files across the Internet between Web servers and browsers.

IDE

See Integrated Development Environment.

*i*FS

See Internet File System.

Integrated Development Environment (IDE)

A set of programs designed to aide in the development of software run from a single user interface. JDeveloper is an IDE for Java development as it includes an editor, compiler, debugger, syntax checker, help system, and so on to permit Java software development through a single user interface.

Internet File System (*i*FS)

The Oracle file system and Java-based development environment that either runs inside the Oracle8i database or on a middle tier and provides a means of creating, storing, and managing multiple types of documents in a single database repository.

Internet Inter-ORB Protocol (IIOP)

The protocol used by CORBA to exchange messages on a TCP/IP network such as the Internet.

instantiate

A term used in object-based languages such as Java and C++ to refer to the creation of an object of a specific class.

*inter*Media

The term used to describe the collection of complex data types and their access within Oracle8*i*. These include text, video, time-series, and spatial data types.

Java

A high-level programming language developed and maintained by Sun Microsystems where applications run in a virtual machine known as a JVM. The JVM is responsible for all interfaces to the operating system. This architecture permits developers to create Java applications and applets that can run on any operating system or platform that has a JVM.

Java Bean

An independent program module that runs within a JVM, typically for creating user interfaces on the client. The server equivalent is called an Enterprise Java Bean (EJB). See also Enterprise Java Bean.

Java Database Connectivity (JDBC)

The programming API that enables Java applications to access a database through the SQL language. JDBC drivers are written in Java for platform independence but are specific to each database.

Java Developer's Kit (JDK)

The collection of Java classes, runtime, compiler, debugger, and usually source code for a version of Java that makes up a Java development environment. JDKs are designated by versions, and Java 2 is used to designate versions from 1.2 onward.

Java Runtime Environment (JRE)

The collection of 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 or application server providing dynamic

functionality to HTML pages. JSPs are actually compiled into servlets when first requested and run in the server's JVM.

Java virtual machine (JVM)

The Java interpreter that converts the compiled Java bytecode into the machine language of the platform and runs it. JVMs can run on a client, in a browser, in a middle tier, on a Web, on an application server such as OAS, or in a database server such as Oracle 8i.

JDBC

See Java Database Connectivity.

JDeveloper

Oracle's Java IDE that enables application, applet, and servlet development and includes an editor, compiler, debugger, syntax checker, help system, etc. In version 3.1,JDeveloper has been enhanced to support XML-based development by including the Oracle XDK for Java integrated for easy use along with XML support in its editor.

JDK

See Java Developer's Kit.

JServer

The Java Virtual Machine that runs within the memory space of the Oracle8i database. In Oracle 8i Release 1 the JVM was Java 1.1 compatible while Release 2 is Java 1.2 compatible.

JVM

See Java virtual machine.

LAN

See local area network.

local area network (LAN)

A computer communication network that serves users within a restricted geographical area. LANs consist of servers, workstations, communications hardware (routers, bridges, network cards, etc.) and a network operating system.

listener

A separate application process that monitors the input process.

Large Object (LOB)

The class of SQL data type that is further divided into Internal LOBs and External LOBs. Internal LOBs include BLOBs, CLOBS, and NCLOBs while External LOBs include BFILES. See also BFILES, Binary Large Object, Character Large Object.

LOB

See Large Object.

namespace

The term to describe a set of related element names or attributes within an XML document. The namespace syntax and its usage is defined by a W3C Recommendation. For example, the <xsl:apply-templates/ > element is identified as part of the XSL namespace. Namespaces are declared in the XML document or DTD before they are used be using the following attribute syntax:- xmlns:xsl="http://www.w3.org/TR/WD-xsl".

NCLOB

See national character Large Object.

mode

In XML, the term used to denote each addressable entity in the DOM tree.

national character Large Object

The LOB datatype whose value is composed of character data corresponding to the database national character set.

NOTATION

In XML, the definition of a content type that is not part of those understood by the parser. These types include audio, video, and other multimedia.

N-tier

The designation for a computer communication network architecture that consists of one or more tiers made up of clients and servers. Typically two-tier systems are made up of one client level and one server level. A three-tier system utilizes two server tiers, typically a database server as one and a Web or application server along with a client tier.

OAG

Open Applications Group.

OAI

Oracle Applications Integrator. Runtime with Oracle iStudio development tool that provides a way for CRM applications to integrate with other ERP systems besides Oracle ERP. Specific APIs must be "message enabled." It uses standard extensibility hooks to generate or parse XML streams exchanged with other application systems. In development.

OAS

See Oracle Application Server.

OASIS

See Organization for the Advancement of Structured Information.

Object View

A tailored presentation of the data contained in one or more object tables or other views. The output of an Object View query is treated as a table. Object Views can be used in most places where a table is used.

object-relational

The term to describe a relational database system that can also store and manipulate higher-order data types, such as text documents, audio, video files, and user-defined objects.

Object Request Broker (ORB)

Software that manages message communication between requesting programs on clients and between objects on servers. ORBs pass the action request and its parameters to the object and return the results back. Common implementations are CORBA and EJBs. See also CORBA.

OE

Oracle Exchange.

Oracle Application Server (OAS)

The Oracle server that integrates all the core services and features required for building, deploying, and managing high-performance, n-tier, transaction-oriented Web applications within an open standards framework.

Oracle Integration Server (OIS)

The Oracle server product that serves as the messaging hub for application integration. OIS contains an Oracle 8i database with AQ and Oracle Workflow and

interfaces to applications using Oracle Message Broker to transport XML-formatted messages between them.

ORACLE_HOME

The operating system environmental variable that identifies the location of the Oracle database installation for use by applications.

OIS

See Oracle Integration Server.

ORB

See Object Request Broker.

Organization for the Advancement of Structured Information (OASIS)

An organization of members chartered with promoting public information standards through conferences, seminars, exhibits, and other educational events. XML is a standard that OASIS is actively promoting as it is doing with SGML.

parent element

An element that surrounds another element, which is referred to as its child element. For example, <Parent><Child></Child></Parent> illustrates a parent element wrapping its child element.

parser

In XML, a software program that accepts as input an XML document and determines whether it is well-formed and, optionally, valid. The Oracle XML Parser supports both SAX and DOM interfaces.

Parsed Character Data (PCDATA)

The element content consisting of text that should be parsed but is not part of a tag or nonparsed data.

PCDATA

See Parsed Character Data.

PDAs

Personal Digital Assistants, such as Palm Pilot.

RDF

Resource Definition Framework.

PL/SQL

The Oracle procedural database language that extends SQL to create programs that can be run within the database.

prolog

The opening part of an XML document containing the XML declaration and any DTD or other declarations needed to process the document.

PUBLIC

The term used to specify the location on the Internet of the reference that follows.

renderer

A software processor that outputs a document in a specified format.

result set

The output of a SQL query consisting of one or more rows of data.

root element

The element that encloses all the other elements in an XML document and is between the optional prolog and epilog. An XML document is only permitted to have one root element.

SAX

See Simple API for XML.

Simple API for XML (SAX)

An XML standard interface provided by XML parsers and used by event-based applications.

schema

The definition of the structure and data types within a database. It can also be used to refer to an XML document that support the XML Schema W3C recommendation.

servlet

A Java application that runs in a server, typically a Web or application server, and performs processing on that server. Servlets are the Java equivalent to CGI scripts.

session

The active connection between two tiers.

SGML

See Structured Generalized Markup Language.

Structured Generalized Markup Language (SGML)

An ISO standard for defining the format of a text document implemented using markup and DTDs.

Structured Query Language (SQL)

The standard language used to access and process data in a relational database.

Server-side Include (SSI)

The HTML command used to place data or other content into a Web page before sending it to the requesting browser.

Secure Sockets Layer (SSL)

The primary security protocol on the Internet, which utilizes a public key / private key form of encryption between browsers and servers.

SQL

See Structured Query Language.

SSI

See Server-side Include.

SSL

See Secure Sockets Layer.

Stylesheet

In XML, the term used to describe an XML document that consists of XSL processing instructions used by an XSL processor to transform or format an input XML document into an output one.

SYSTEM

The term used to specify the location on the host operating system of the reference that follows.

tag

A single piece of XML markup that delimits the start or end of an element. Tags start with < and end with >. In XML, there are start-tags (<name>), end-tags (</name>), and empty tags (<name/>).

TCP/IP

See Transmission Control Protocol/Internet Protocol.

thread

In programming, a single message or process execution path within an operating system that supports multiple operating systems, such as Windows, UNIX, and Java.

Transmission Control Protocol/Internet Protocol (TCP/IP)

The communications network protocol that consists of the TCP which controls the transport functions and IP which provides the routing mechanism. It is the standard for Internet communications.

Transviewer

The Oracle term used to describe the Oracle XML Java Beans included in the XDK for Java. These beans include an XML Source View Bean, Tree View Bean, DOMParser Bean, Transformer Bean, and a TransViewer Bean.

user interface (UI)

The combination of menus, screens, keyboard commands, mouse clicks, and command language that defines how a user interacts with a software application.

Uniform Resource Identifier (URI)

The address syntax that is used to create URLs and 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.

URI

See Uniform Resource Identifier.

URL

See Uniform Resource Locator.

valid

The term used to refer to an XML document when its structure and element content is consistent with that declared in its referenced or included DTD.

W3C

See World Wide Web Consortium (W3C).

WAN

See wide area network.

Web Request Broker (WRB)

The cartridge within OAS that processes URLs and sends them to the appropriate cartridge.

well-formed

The term used to refer to an XML document that conforms to the syntax of the XML version declared in its XML declaration. This includes having a single root element, properly nested tags, and so forth.

wide area network (WAN)

A computer communication network that serves users within a wide geographic area, such as a state or country. WANs consist of servers, workstations, communications hardware (routers, bridges, network cards, etc.), and a network operating system.

Working Group (WG)

The committee within the W3C that is made up of industry members that implement the recommendation process in specific Internet technology areas.

World Wide Web Consortium (W3C)

An international industry consortium started in 1994 to develop standards for the World Wide Web. It is located at www.w3c.org.

Wrapper

The term describing a data structure or software that wraps around other data or software, typically to provide a generic or object interface.

XML Developer's Kit (XDK)

The set of libraries, components and utilities that provide software developers with the standards-based functionality to XML-enable their applications. In the case of the Oracle XDK for Java, the kit contains an XML Parser, XSL Processor, XML Class Generator, the Transviewer Java Beans and the XSQL Servlet.

XLink

The XML Linking language consisting of the rules governing the use of hyperlinks in XML documents. These rules are being developed by the XML Linking Group under the W3C recommendation process. This is one of the three languages XML supports to manage document presentation and hyperlinks (XLink, XPointer, and XPath).

XML

See eXtensible Stylesheet Language.

XML query

The W3C's effort to create a standard for the language and syntax to query XML documents.

XML schema

The W3C's effort to create a standard to express simple data types and complex structures within an XML document. It addresses areas currently lacking in DTDs, including the definition and validation of data types. Oracle XML Schema Processor automatically ensures validity of XML documents and data used in e-business applications, including online exchanges. It adds simple and complex datatypes to XML documents and replaces DTD functionality with an XML Schema definition XML document.

XPath

The open standard syntax for addressing elements within a document used by XSL and XPointer. XPath is currently a W3C recommendation. It specifies the data model and grammar for navigating an XML document utilized by XSLT, XLink and XML Query.

XPointer

The term and W3C recommendation to describe a reference to an XML document fragment. An XPointer can be used at the end of an XPath-formatted URI. It specifies the identification of individual entities or fragments within an XML document using XPath navigation.

XSL

See eXtensible Stylesheet Language.

XSLFO

See eXtensible Stylesheet Language Formatting Object.

XSLT

See eXtensible Stylesheet Language Transformation.

XSQL

The designation used by the Oracle Servlet providing the ability to produce dynamic XML documents from one or more SQL queries and optionally transform the document in the server using an XSL stylesheet.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X