Oracle9*i*

Case Studies - XML Applications

Release 1 (9.0.1)

June, 2001

Part No. A88895-01

ORACLE®

Oracle9*i* Case Studies - XML Applications, Release 1 (9.0.1)

Part No. A88895-01

Primary Author: Shelley Higgins

Graphics: Valerie Moore

Contributing Authors: Sandeepan Banerjee, Robert Dell’immagine, Robert Hall, Karun K, Murali Krishnaprasad, Olivier LeDiouris, Paul Nock, Ami Parekh, Rajesh Raheja, Carol Roston, Frank Rovitto, Mark Scardina, Manh-Kiet (Allen) Yap

Contents

Send Us Your Comments

Oracle9*i* Case Studies - XML Applications, Release 1 (9.0.1)

Part No. A88895-01

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

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

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

- Electronic mail: infodev_us@.oracle.com
- FAX: (650) 506-7227   Attn: Server Technologies Documentation Manager
- Postal service:
  Oracle Corporation
  Server Technologies Documentation
  500 Oracle Parkway, Mailstop 4op11
  Redwood City, 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 provides case studies and applications that use Oracle9*i*'s XML-enabled database technology. It describes different ways that XML data can be stored, managed, queried, and exchanged in the database using Oracle XML-enabled technology.

This manual describes several scenarios that are based on actual business applications. The case studies are presented according to their main function, namely, whether they are primarily used for one or both of the following high level tasks:

- XML-Based Content and / or Document Management, see Part II, "Managing Content and Documents with XML".
- XML Data Exchange in Business--to-Business (B2B), Business-to-Consumer (B2C), Application-to-Application (A2A), or Peer-to-Peer(P2P) applications. See Part III, "XML Data Exchange". A detailed application is described in, Chapter 8, "Online B2B XML Application: Step by Step". This explains how to build an XML B2B data exchange and customized presentation application.

Composed or Decomposed (Generated) XML

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

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

Oracle XML-Enabled Technology

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

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

See:

- *Oracle9i Application Developer's Guide - XML*
- *Oracle9i XML Reference*

This is the first edition of this manual. A number of helpful chapters, waiting in the wings, did not make it into this edition. If you have, or know of any interesting XML Oracle database case studies that you would like to contribute, or would like to see included here, please inform the author through infodev_us@oracle.com.

Audience

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

Prerequisite Knowledge

An understanding of XML and XSL is helpful but not essential for using this manual. References to good sources for more information are included in Appendix A, "An XML Primer", and in the FAQ section at the end of Chapter 3 in *Oracle9i Application Developer's Guide - XML*.

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

Feature Coverage and Availability

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

How this Manual is Organized

This manual is organized into 3 parts and 9 chapters. It includes an appendix, index and glossary.

- PART 1 Introducing Oracle XML-Enabled Technology.
  - Chapter 1, "Oracle XML-Enabled Technology" has introductory and basic information about using Oracle9i's XML components, XML support in the database, using XMLType and URI-Reference, XML SQL

Utility (XSU), and how to apply Oracle Text to search and retrieve information from XML documents.

  * Chapter 2, "Modeling and Design Issues for Oracle XML Applications" introduces you to the various ways that XML can be stored in the database, XML design issues, and loading XML into the database. It also presents some actual business scenarios and describes the XDK and other Oracle XMl enabled components you can consider using.

- PART II Managing Content and Documents with XML

  * Chapter 3, "Oracle9i AS Wireless Edition and XML" describes Wireless Edition (portal-to-go) components an how they are used to extract content from a web site, convert this to XML, and transform the data for display on any device.

  * Chapter 4, "Customizing Presentation with XML and XSQL: Flight Finder" describes how the Flight Finder application generates XML to and from the database and uses XSQL Servlet to process queries and output results as XML. It also discusses how Flight Finder formats the XML data using XSL stylesheets. This application and demo is also available on Oracle Technology Network (OTN).

  * Chapter 5, "Customizing Content with XML: Dynamic News Application" describes the Dynamic News application, the three custom servlets used in the application, and how XML SQL Utility (XSU) accesses news content from Oracle9i. The application offers three levels of user customization — static, semi-dynamic, and dynamic. This chapter also includes some detail on customizing presentations.

  * Chapter 6, "Using Oracle9i Internet File System (9iFS) to Build XML Applications" introduces you to the 9iFS and focuses on 9iFS's XML support.

- PART III XML Data Exchange

  * Chapter 7, "Customizing Discoverer 4i Viewer with XSL" describes how Discoverer4i (9i) can be used to customize web applications to make the best use of your business intelligence, forms, and reports. The chapter includes some detail on using XSL stylesheets to customize your presentation.

  * Chapter 8, "Online B2B XML Application: Step by Step" describes in detail one way to build and implement a B2B online catalog XML application using XSQL Servlet and Advanced Queueing (AQ). It includes scripts for transforming the resulting XML messages for

different user devices. An extension and simplified version of this application is also available at http://http:otn.oracle.com/tech/xml

* Chapter 9, "Service Delivery Platform (SDP) and XML" introduces you to the Phone Number Portability application. The chapter summarizes how XML messaging can be used by various telecommunications products and systems and designed in *i*Message Studio, Event Manager, and Adapters.

Related Documentation

For more information, see these Oracle resources:

- *Oracle9i Application Developer's Guide - XML*
- *Oracle9i New Features* for information about the differences between Oracle9*i* and the Oracle9*i* Enterprise Edition and the available features and options. That book also describes all the features that are new in Oracle9*i*.
- *The JDeveloper Guide*
- *Oracle9i Application Developer's Guide - Fundamentals*
- *Oracle8i Application Developer's Guide - Advanced Queuing*
- *Oracle9i Supplied PL/SQL Packages Reference*
- *Oracle Integration Server Overview*
- *Oracle9i XML Reference*

How to Order this Manual

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

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

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

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

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

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

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

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

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

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

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

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

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

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

How to Access this Manual On-Line

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

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

Conventions

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

- Conventions in Text
- Conventions in Code Examples

Conventions in Text

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

Convention	Meaning	Example
Bold	Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.	When you specify this clause, you create an **index-organized table**.
Italics	Italic typeface indicates book titles or emphasis.	*Oracle9i Concepts* Ensure that the recovery catalog and target database do *not* reside on the same disk.
`UPPERCASE monospace (fixed-width font)`	Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, usernames, and roles.	You can specify this clause only for a `NUMBER` column. You can back up the database by using the `BACKUP` command. Query the `TABLE_NAME` column in the `USER_TABLES` data dictionary view. Use the `DBMS_STATS.GENERATE_STATS` procedure.
`lowercase monospace (fixed-width font)`	Lowercase monospace typeface indicates executables, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, usernames and roles, program units, and parameter values. **Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	Enter `sqlplus` to open SQL*Plus. The password is specified in the `orapwd` file. Back up the datafiles and control files in the `/disk1/oracle/dbs` directory. The `department_id`, `department_name`, and `location_id` columns are in the `hr.departments` table. Set the `QUERY_REWRITE_ENABLED` initialization parameter to `true.` Connect as `oe` user. The `JRepUtil` class implements these methods.

Convention	Meaning	Example
`lowercase monospace (fixed-width font) italic`	Lowercase monospace italic font represents placeholders or variables.	You can specify the `parallel_clause`. Run `Uold_release.SQL` where `old_release` refers to the release you installed prior to upgrading.

Conventions in Code Examples

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

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

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

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

|	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	`{ENABLE | DISABLE}`
		`[COMPRESS | NOCOMPRESS]`

...	Horizontal ellipsis points indicate either:
	■ That we have omitted parts of the code that are not directly related to the example
	■ That you can repeat a portion of the code
		
		`CREATE TABLE ... AS subquery;`
		`SELECT col1, col2, ... , coln FROM employees;`

. . .	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	

Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown.	`acctbal NUMBER(11,2);`
		`acct    CONSTANT NUMBER(4) := 3;`

Convention	Meaning	Example

`Italics`	Italicized text indicates placeholders or variables for which you must supply particular values.	`CONNECT SYSTEM/`*`system_password`*
		`DB_NAME =` *`database_name`*

`UPPERCASE`	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase.	`SELECT last_name, employee_id FROM employees;`
		`SELECT * FROM USER_TABLES;`
		`DROP TABLE hr.employees;`

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

Documentation Accessibility

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

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

Reading Code Examples

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

What's New in Oracle XML-Enabled Technology?

This section describes the new features in the following releases:

- **XML Features Introduced with Oracle9i, Release 1 (9.0.1)**
- **XML Features Introduced with Oracle8i Release 3 (8.1.7)**

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

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

XDK for Java

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

See: *Oracle9i Application Developer's Guide - XML*, "Using XML Schema Processor for Java

- **XSQL**
  - Support for Database Bind Variables. Now both lexical substitution and true database bind variables are supported for improved performance.

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

See Also: *Oracle9i Application Developer's Guide - XML,* "XSQL Pages Publishing Framework"

- **XDK for Java Beans**
  - DBViewer Bean (new). Displays database queries or any XML by applying XSL stylesheets and visualizing the resulting HTML in a scrollable swing panel.

- DBAccess Bean (new). DB Access bean maintains CLOB tables that hold multiple XML and text documents.

  See: *Oracle9i Application Developer's Guide - XML*, "Using XML Transviewer Beans"

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

  See: *Oracle9i Application Developer's Guide - XML*, "Using XML Schema Processor for C"

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

  See: *Oracle9i Application Developer's Guide - XML*,,"Using XML Schema Processor for C++"

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

XML SQL Utility (XSU) Features

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

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

See: *Oracle9i Application Developer's Guide - XML*, "XML SQL Utility (XSU)"

Database XML Related Enhancements

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

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

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

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

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

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

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

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

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

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

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

  See: *Oracle9i Application Developer's Guide - XML*,

  - "Database Support for XML"
  - "Database Uri-references"
  - "Searching XML Data with Oracle Text"

Advanced Queueing (AQ) Features

New Advanced Queueing features include enhanced XML messaging options:

- Internet-Data-Access-Presentation (IDAP)

- AQXMLServlet for use with HTTP and SMTP access
- XMLType Queues
- XML AQ message transformation

See: *Oracle9i Application Developer's Guide - XML*, "Exchanging XML Data Using Oracle AQ"

Metadata API

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

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

Metadata API is available on Oracle9*i* whenever the instance is operational. It is not available on Oracle Lite. It includes the (new) DBMS_METADATA PL/SQL supplied package.

See: *Oracle9i Application Developer's Guide - XML*, "Using Metadata API".

Oracle Text *(inter*Media Text/Context) Features

The new section group, PATH_SECTION_GROUP, enables new and more sophisticated section searching for XML documents:

The new Oracle Text operators are:

  - HASPATH() operator
  - INPATH() operator

Oracle Text's PATH_SECTION_GROUP features include the following support:

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

See: *Oracle9i Application Developer's Guide - XML*, "Searching XML Data with Oracle Text"

Phone Number Portability (SDP)

- Expanded support for new applications including ADSL and wireless

  See: Chapter 9, "Service Delivery Platform (SDP) and XML"

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

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

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

Part I

Introducing Oracle XML-Enabled Technology

Part I of the book introduces you to Oracle XML-enabled technology and features. It contains the following chapters:

1 Oracle XML-Enabled Technology

This chapter describes the following sections:

What is XML ?

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

What are Oracle XML-Enabled Technologies?

XML models structured and semi-structured data. Oracle9*i* supports structured and semi-structured data, as well as complex and unstructured data. Oracle9i is XML-enabled in that it natively handles the storage, query, presentation, and manipulation of XML data.

Oracle XML Components

Oracle XML components are comprised of the following:

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

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

Some typical XML-based business solution are:

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

– Customized presentation. Dynamic News case study, Portal-to-Go, and Flight Finder

Oracle Development Tools and Frameworks

XSQL Servlet and Pages, Oracle9i Internet File System (9*i*FS), JDeveloper, Business Components for Java (BC4J), Oracle Portal (WebDB), Oracle9iAS Reports Services, and Oracle9i Dynamic Services can all be used to build XML applications.

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

Database and Middle Tier

XML applications can either reside on the database or on a middle tier, such as Oracle9*i* Application Server, Apache Server, or other Java enabled Web servers.

Data Stored in the Database

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

Storing and Retrieving XML Data from Oracle9*i*

XML has emerged as the standard for data interchange on the Web and Oracle9*i* is XML-enabled to natively store, search, and retrieve XML in the following formats:

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

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

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

XML Support in the Database

Oracle9*i* provides the following XML support:

- *Generation of XML Documents*. Oracle9*i* supports the generation of XML on the client or server, where existing object-relational data can be used to generate the corresponding XML. XML can be generated from query results or as part of the SQL query itself using the XSU (XML SQL Utility) Java or PL/SQL API.

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

  - By providing new SQL functions for XML generation and aggregation
  - By providing a C version of the XML SQL Utility, linked to the server

- *Storage, Querying, and Retrieval of XML documents*. Before this release you could use, for example, XSU to store, query, and retrieve XML documents. Now, with this release you can use the new datatype, XMLType.

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

  See: *Oracle9i Case Studies - XML Applications*:, "Database Support for XML", under "Indexing XMLType columns".

XML and URI Data Types

Oracle9*i* provides new types to handle XML and URI data. The Extensible Markup Language (XML) is a standard format developed by the World Wide Web Consortium (W3C) for representing structured and un-structured data on the Web.

URIs or Universal Resource Identifiers are used to identify resources such as web pages anywhere on the web. Oracle9*i* provides a new class of URIs to access data stored in the database itself, called *DBUri-refs*. It also provides a new set of types to store and access both external and internal URIs from the database.

XMLType

The Oracle supplied type, `XMLType`, can be used to store and query XML data in the database. `XMLType` provides member functions to access, extract and query the XML data using XPath expressions. XPath is another standard developed by the W3C committee to traverse XML documents. In Oracle9i, `XMLType` functions only support a limited subset of the XPath expressions. Oracle9i also provides a set of

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

> **See:** *Oracle9i Application Developer's Guide - XML*
>
> - "Database Support for XML"
> - "Searching XML Data with Oracle Text"
> - "Exchanging XML Data Using Oracle AQ"
> - *Oracle9i Application Developer's Guide - Advanced Queuing*, or information about using XMLType with Oracle Advanced Queuing

URI Data Types

Oracle9*i* supplies the following family of Uri types:

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

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

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

Since `UriType` is the super type, you can create columns of this type and store `DBUriType` or `HttpUriType` instances in this column. This allows you to reference data stored inside or outside the database and access them consistently. DBUri-ref uses an XPath like representation to reference data inside the database. Using this, you can reference data stored in CLOBs or other columns and expose them as URLs to the external world. Oracle9*i* provides a standard servlet than can be installed and run under the Oracle Servlet engine which can interpret such URLs.

See: *Oracle9i Application Developer's Guide - XML*, "Database Uri-references"

Extensibility and XML

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

Oracle Text Searching

XML text stored in LOBs can be indexed using the extensibility indexing interface. Oracle9*i* provides operators such as `CONTAINS` and `WITHIN` that you can use to search within the XML text for substring matches.

See: *Oracle9i Application Developer's Guide - XML* "Searching XML Data with Oracle Text"

Oracle-Based XML Applications

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

Content and Document Management

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

See: Content and Document Management Chapters in *Oracle9i Case Studies - XML Applications*:

- "Customizing Content with XML: Dynamic News Application"
- "Oracle9i AS Wireless Edition and XML"
- "Customizing Presentation with XML and XSQL: Flight Finder"

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

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

See: The following chapters in this manual:

- "Exchanging XML Data Using Oracle AQ"
- "Customizing Discoverer 4i Viewer with XSL"
- "Service Delivery Platform (SDP) and XML"
- "How Oracle Exchange Uses XML"
- "B2B XML Application: Step by Step"

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

Oracle XML-Enabled Technology Components and Features

Oracle9*i* is well-suited for building XML database applications. Oracle XML-enabled technology has the following features:

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

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

Oracle 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 as large as 4 Gigabytes each stored in a column in a table.

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

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

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

See Also: ■ *Oracle9i Text Reference*

Messaging Hubs and Middle Tier Components

Also included in Oracle XML are the following components:

- *XML-Enabled Messaging Hubs*. These hubs are vital in business-to-business applications that interface with non-Oracle systems.
- *Middle Tier Systems*: XML-enabled application, web, or integrated servers, such as Oracle9*i* Application Server.

Oracle JVM (Java Virtual Machine)

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

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

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

Oracle9*i* Application Server

Oracle9*i* Application Server (Oracle9iAS), offers services for both intranet and internet Web applications. It is integrated with Oracle9*i* and offers advanced services such as data caching and Oracle Portal. Oracle9iAS also provides other services including Oracle Advanced Queueing, Oracle Message Broker, Oracle Workflow, Oracle9i Reports Services, Dynamic Services, and more.

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

Back-End to Database to Front-End Integration Issues

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

Such data exchange between different vendors' relational and object-relational databases is simpler using XML. One example of a data exchange implementation using XML and AQ is provided in ______.

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

Higher Performance Implications

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

- Processing database data and XML together on the same server helps eliminate network traffic for data access.
- Exploiting the speed of the Oracle9i query engine and Oracle JVM, Oracle9i Application Server, or OIS further enhances data access speed.
- XDK for C components can be used for their native XML capabilities and higher performance

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

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

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

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

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

For more information, see The following chapters in the manual, *Oracle9i Application Developer's Guide - XML*.

Writing Custom XML Applications

Writing custom applications that process XML documents can be simpler in an Oracle9*i* 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 Oracle9*i* platform on every operating system where Oracle9*i* is ported.

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

The Oracle Suite of Integrated Tools and Components

Oracle9*i* provides an integrated suite of tools and components for building e-business applications:

- Oracle JDeveloper and Oracle Business Components for Java (BC4J)
- Oracle9i Internet File System (Oracle 9iFS or 9iFS)
- Oracle Portal
- Oracle Exchange

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

Oracle JDeveloper and Oracle Business Components for Java (BC4J)

Oracle JDeveloper is an integrated environment for building, deploying, and debugging applications leveraging Java and XML on Oracle9*i*. It facilitates working in Java 1.1 or 1.2 with CORBA, EJB, and Java Stored Procedures. With it you can do the following:

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

Examples of applications built using Oracle JDeveloper include:

- *i*Procurement (Self Service Applications) including Self-Service Web-Expensing.
- Content Delivery for PDAs. See the chapter, "Oracle9i AS Wireless Edition and XML".
- Online Marketplaces
- Retailer - Supplier transaction using XML and AQ messaging. See "B2B XML Application: Step by Step".

See "Using JDeveloper to Build Oracle XML Applications" in *Oracle9i Application Developer's Guide - XML*, for more information on using JDeveloper to build XML applications.

Oracle Business Components for Java (BC4J) Business Components for Java (BC4J) is an Oracle9*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.

Note: Oracle JDeveloper and BC4J are not included with Oracle9i. Only the BC4J runtime is included. You can download JDeveloper from OTN.

Oracle9i Internet File System (Oracle 9iFS or 9iFS)

Access to Oracle9*i* Internet File System (9*i*FS) facilitates organizing and accessing documents and data using a file- and folder-based model through standard Windows and Internet protocols such as SMB, HTTP, FTP, SMTP, and IMAP4.

9*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 file, into Oracle9*i* and display the document from a Web server, such as Oracle9*i* Application Server or Apache Web Server.

See Also: Using Internet File System (iFS) to Build XML Applications"

9*i*FS is a simple way for developers to work with XML, where *i*FS serves as the repository for XML. 9*i*FS automatically parses XML and stores content in tables and columns. 9*i*FS renders the content when a file is requested delivering select information, for example, on the Web.

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

Oracle Portal

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

Oracle Portal is software for building and deploying enterprise portals, the Web sites that power an e-business. The browser interface delivers an organized, personalized view of business information, Web content, and applications needed by each user. It includes site-building and self-service Web publishing functionality of WebDB 2.2 and adds new enterprise portal features such as single sign-on, personalization, and content classification. Oracle Portal uses Oracle9*i* and is deployed on and packaged with Oracle9i Application Server.

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

> **See Also:** *Oracle9i Application Developer's Guide - XML*, "Using the PDK for Visualizing XML Data in Oracle Portal" for an introduction to Oracle Portal's PDF and URL Services.

Oracle Exchange

The Oracle Exchange platform is based on Oracle9*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 ongoing service and support.

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

XML Gateway

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

Metadata API

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

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

Metadata API is available on Oracle9*i* whenever the instance is operational. It is not available on Oracle Lite.

See Also: *Oracle9i Application Developer's Guide - XML*, "Using Metadata API"

Other XML Initiatives

Besides these tools, the following initiatives are underway.

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

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

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

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

See: Oracle9i Application Developer's Guide - XML, Exchanging XML Data Using Oracle AQ"

Oracle XML Samples and Demos

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

What Is Needed to Run Oracle XML Components

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

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

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

Requirements for XDK

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

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

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

Which XML Components are Included with Oracle9i Database and Oracle9i Application Server?

Table 1–1 lists the XDK component versions included with Oracle9*i* Database and Oracle9*i* Application Server (Oracle9*i*AS):

Table 1–1 Oracle9i and Oracle9iAS XDK Component Supplied Versions

XDK Component	Oracle9*i* Database Rel. 1(9.0.1)	Oracle9*i*AS Rel.--prellimary version nos. only
XDK for Java		
XML Parser for Java and XSLT Processor	9.0.1.0.0	9.0.1.0.0
XML Schema Processor for Java	9.0.1.0.0	9.0.1.0.0
XML Class Generator for Java	9.0.1.0.0	9.0.1.0.0
XSQL Servlet	9.0.1.0.0	9.0.1.0.0
XML SQL Utility (XSU) for Java	9.0.1.0.0	9.0.1.0.0
XDK for Java Beans		
XML Transviewer Beans	9.0.1.0.0	9.0.1.0.0
XDK for C		
XML Parser for C and XSLT Processor	9.0.1.0.0	9.0.1.0.0
XML Schema Processor for C	9.0.1.0.0	9.0.1.0.0
XDK for C++		
XML Parser for C++ and XSLT Processor	9.0.1.0.0	9.0.1.0.0
XML Schema Processor for C++	9.0.1.0.0	9.0.1.0.0
XML Class Generator for C++	9.0.1.0.0	9.0.1.0.0
XDK for PL/SQL		
XML Parser for PL/SQL and XSLT Processor	9.0.1.0.0	9.0.1.0.0
XML SQL Utility (XSU) for PL/SQL	9.0.1.0.0	9.0.1.0.0

XML Technical Support

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

```
http://otn.oracle.com/tech/xml
```

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

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

Download the Latest Software From OTN

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

```
http://otn.oracle.com/software/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.

2

Modeling and Design Issues for Oracle XML Applications

This chapter contains the following sections:

XML Data can be Stored as Generated XML or Composed XML

XML data can be stored in Oracle9*i* in the following ways:

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

Generated XML

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

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

Generated XML Examples

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

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

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

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

Object-Relational Storage for Generated XML Documents

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

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

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

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

> **See:** *Oracle9i Application Developer's Guide - XML*, "XML SQL Utility (XSU)"

Composed (Authored/Native) XML

Oracle8*i* and higher support the storage of large objects or LOBs as character LOBs (CLOB), binary LOBs (BLOB), or externally stored binary files (BFILE). LOBs are used to store composed (Authored / Native) XML documents.

Storing Composed XML Data in CLOBs or BFILEs

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

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

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

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

- Composed XML examples include written text such as articles, advertisements, books, legal contracts, and so on. Documents of this nature are known as document-centric and are delivered from the database as a whole. Storing this kind of document intact within Oracle gives you the advantages of an industry-proven database and its reliability over file system storage.

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

Oracle Text *(inter*Media Text) Indexing Enables Fine Grain Searching of XML Element Content

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

Oracle8*i* and Oracle9*i* recognize XML tags, and section and sub-section text searching within XML elements' content. The result is that queries can be posed on unstructured data and restricted to certain sections or elements within a document.

Oracle Text Example: Searching Text and XML Data Using CONTAINS

This Oracle Text (*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: *Oracle9i Application Developer's Guide - XML*, "Searching XML Data with Oracle Text", for more information on Oracle Text.

Advantages of Using Composed (Authored) XML Storage

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

Disadvantages of Using Composed XML Storage

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

Using a Hybrid XML Storage Approach for Better Mapping Granularity

The previous section described the following:

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

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

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

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

A Hybrid Approach Allows for User-Defined Storage Granularity

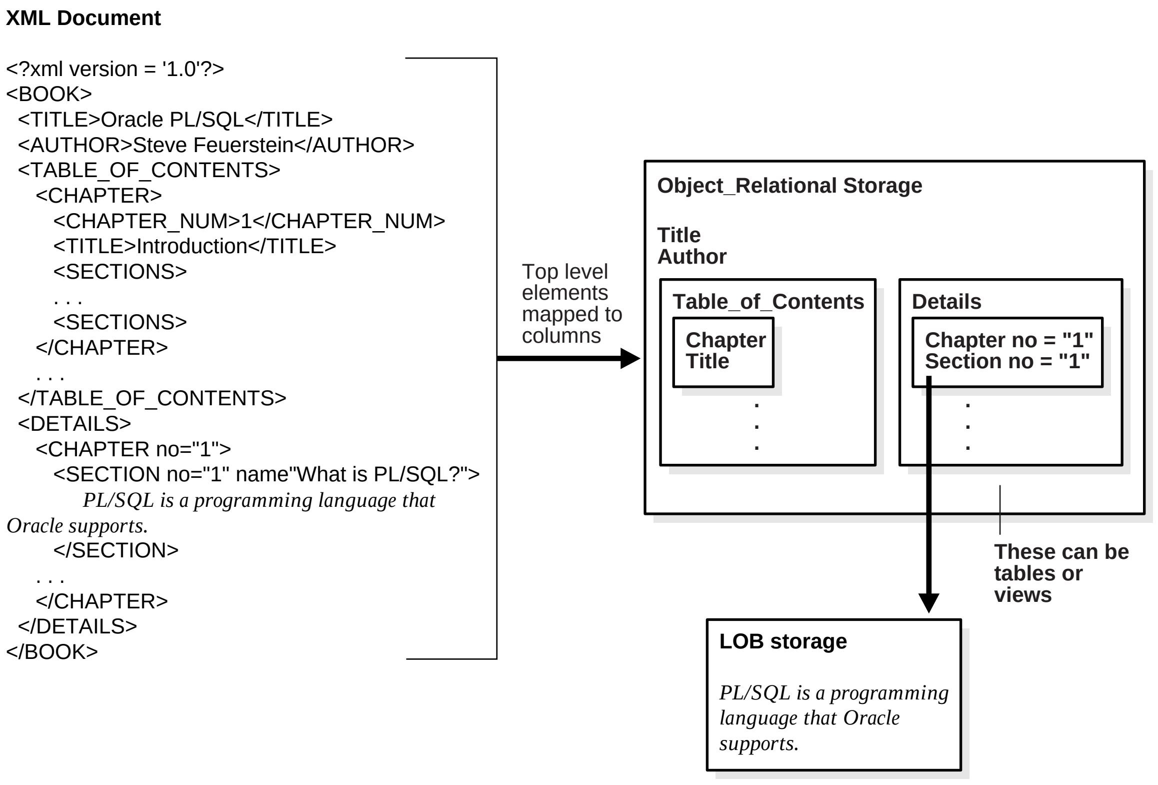
The alternative is to have user-defined granularity for such storage. In the book example, you may want the following:

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

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

Figure 2–1 illustrates this hybrid approach to XML storage.

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


XML Document
<?xml version = '1.0'?>
<BOOK>
<TITLE>Oracle PL/SQL</TITLE>
<AUTHOR>Steve Feuerstein</AUTHOR>
<TABLE_OF_CONTENTS>
<CHAPTER>
<CHAPTER_NUM>1</CHAPTER_NUM>
<TITLE>Introduction</TITLE>
<SECTIONS>
. . .
<SECTIONS>
</CHAPTER>
. . .
</TABLE_OF_CONTENTS>
<DETAILS>
<CHAPTER no="1">
<SECTION no="1" name"What is PL/SQL?">
PL/SQL is a programming language that Oracle supports.
</SECTION>
. . .
</CHAPTER>
</DETAILS>
</BOOK>
Top level elements mapped to columns
Object_Relational Storage
Title
Author
Table_of_Contents
Chapter
Title
Details
Chapter no = "1"
Section no = "1"
These can be tables or views
LOB storage
PL/SQL is a programming language that Oracle supports.

Hybrid Storage Advantages

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

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

Transforming Generated XML

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

When the XML Document Structure Needs Transforming

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

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

Combining XML Documents and Data Using Views

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

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

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

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

Using XSLT to Transform Query Results

This involves querying on the original document and transforming the result into a form required by the 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 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.

Indexing and Querying Transformations

You may need to create indexes and query on transformed views of an XML document. For example, in an XML messaging environment, there could be purchase order messages in different formats. You may want to query them canonically, so that a particular query can work across all purchase order messages.

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

Indexing Approaches

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

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

See Also:

- *Oracle9i Application Developer's Guide - XML*, "Searching XML Data with Oracle Text", for more information on Oracle Text.
- *Oracle9i Text Developer's Guide*

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

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

XML Schemas and Mapping of Documents

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

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

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

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

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

XMLSchema Example 1: Defining a Simple Data Type

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

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

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

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

XMLSchema Example 2: Using XMLSchema 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"XML Example 2: XML Document Produced from Generic Mapping". The point is to underscore the closeness of match between the proposed constructs of XML Schema with SQL:1999-based type systems. Given such a close match, it is relatively easy to map an XML Schema to a database Object-Relational schema, and map documents that arevalid according to the above schema to row objects in the database schema. In fact, the greater expressiveness of XML Schema over DTDs greatly facilitates the mapping.

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

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

General XML: Design Issues for Data Exchange Applications

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

- Generating a Web Form from XML Data Stored in the Database
- Sending XML Data from a Web Form to the Database

Generating a Web Form from XML Data Stored in the Database

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

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

Sending XML Data from a Web Form to the Database

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

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

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

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

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

Sending XML Documents Applications-to-Application

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

Here you can assume the following:

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

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

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

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

Advanced Queuing. An Oracle database sends an XML document via Net Services, HTTP or SMTP, and JDBC to the one or more receiving applications as a message through Oracle Advanced Queueing (AQ). The receiving applications dequeue the XML message and process it.

See Also:

- *Oracle9i Application Developer's Guide - XML*, "Exchanging XML Data Using Oracle AQ"
- Chapter 8, "Online B2B XML Application: Step by Step"
- *Oracle9i Application Developer's Guide - Advanced Queuing*

Loading XML into a Database

You can use the following options to load XML data or DTD files into Oracle9*i*:

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

You can also use Oracle9i Internet File System (9*iFS*) to put an XML document into the database. However, it does not support DTDs. It does however support XML Schema, the standard that will replace DTDs.

Using SQL*Loader

You can use SQL*Loader to bulk load LOBs.

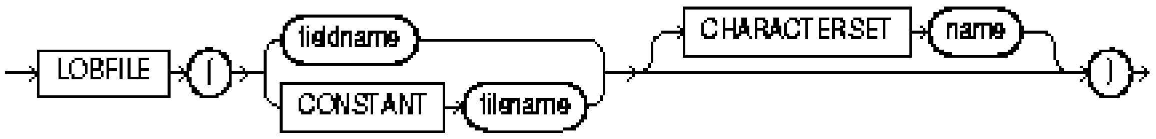
See:

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

Loading XML Documents Into LOBs With SQL*Loader

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

Figure 2–2 The LOBFILE Syntax



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

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

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

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

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

A direct-path load does not compete with other users for database resources, so it can usually load data at near disk speed. Considerations inherent to direct path loads, such as restrictions, security, and backup implications, are discussed in Chapter 9 of *Oracle9i Utilities*.

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

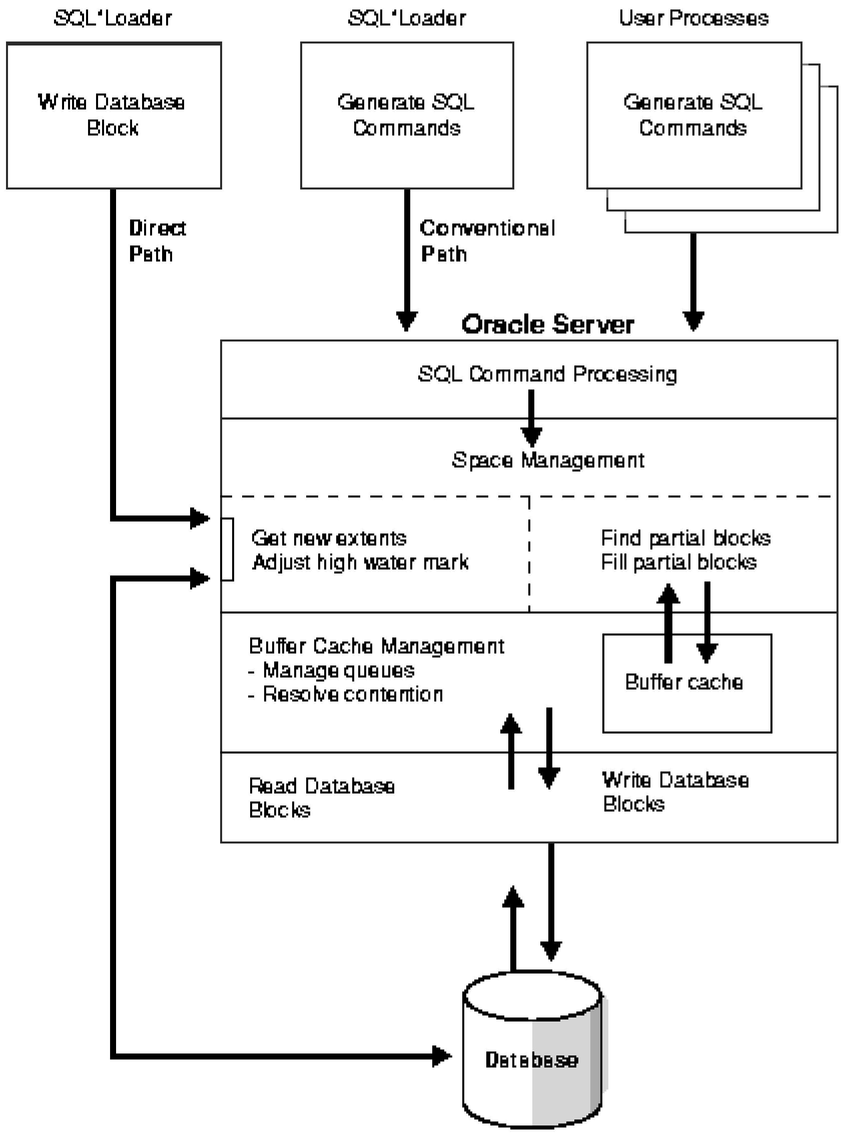
Tables to be loaded must already exist in the database. SQL*Loader never creates tables. It loads existing tables that either already contain data or are empty.

The following privileges are required for a load:

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

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

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



Applications that Use Oracle XML -EnabledTechnology

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

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

or any combinations of these. This manual focuses on these two application areas, in Part II, "Managing Content and Documents with XML" and Part III, "XML Data Exchange", respectively.

Business applications scenarios for each of these two areas are described later in this chapter.

Content and Document Management with Oracle XML-Enabled Technology

Customizing Presentation of Data

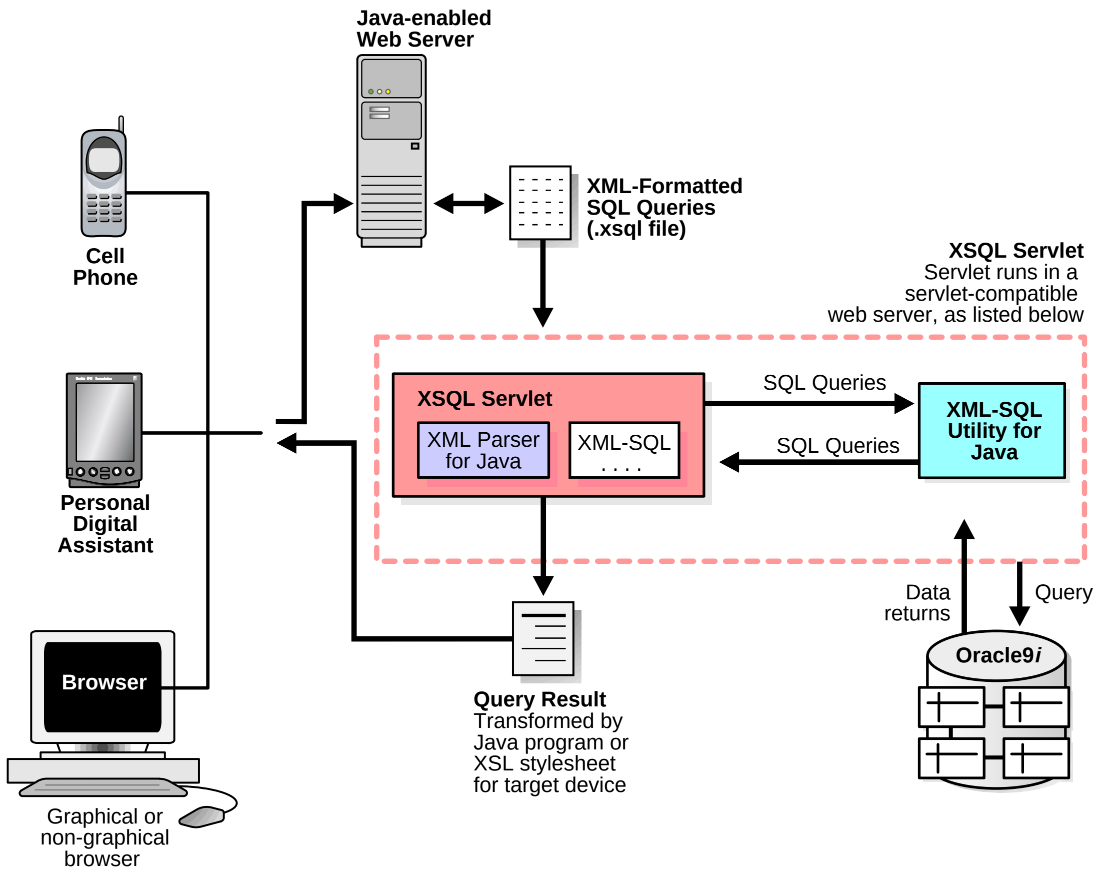
XML is increasingly used to enable customized presentation of data for different browsers, devices, and users. By using XML documents along with XSL stylesheets on either the client, middle-tier, or server, you can transform, organize, and present XML data tailored to individual users for a variety of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

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

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

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

Figure 2–4 Content Management: Customizing Your Presentation


Java-enabled Web Server
XML-Formatted SQL Queries (.xsql file)
XSQL Servlet Servlet runs in a servlet-compatible web server, as listed below
Cell Phone
Personal Digital Assistant
XSQL Servlet
XML Parser for Java
XML-SQL . . . .
SQL Queries
SQL Queries
XML-SQL Utility for Java
Data returns
Query
Oracle9*i*
Browser
Graphical or non-graphical browser
Query Result Transformed by Java program or XSL stylesheet for target device

See Also: Oracle9i Application Developer's Guide - XML,"Using XML Parser for Java"

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

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

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

These scenarios are illustrated with case studies in Part II, "Managing Content and Documents with XML"

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

Problem

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

Solution

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

Use XMLType to store the documents in XML format, where the relational data is updatable. Use Oracle9*i*'s Internet File System (9*i*FS) as the data repository interface. 9*i*FS helps implement XML data repository management and administration tasks.

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

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

See Figure 2–5.

Main Tasks Involved

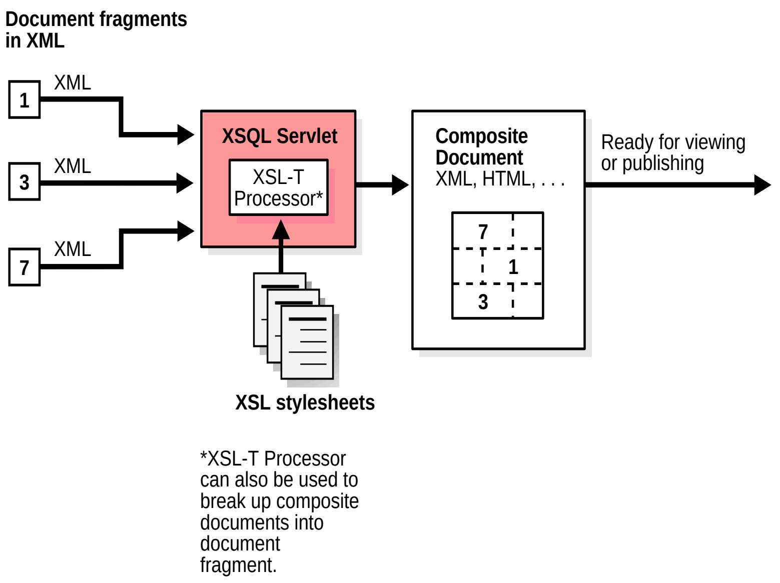
These are the main tasks involved in Scenario 1's solution:

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

Oracle XML Components Used

- XML Parser with XSLT
- XSQL Servlet and XSU to move sections or fragments into and out of the database

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


Document fragments in XML
1
XML
3
XML
7
XML
XSQL Servlet
XSL-T Processor*
XSL stylesheets
Composite Document
XML, HTML, . . .
7
1
3
Ready for viewing or publishing
*XSL-T Processor can also be used to break up composite documents into document fragment.

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

Problem

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

Solution

Use XSL stylesheets with the XSQL Servlet to dynamically deliver appropriate rendering to the requesting service. See Figure 2–6. See also, Oracle9iAS Dynamic Services and Oracle Syndication Server (OSS) chapters in *Oracle9i Application Developer's Guide - XML*.

See Also:

- Chapter 3, "Oracle9i AS Wireless Edition and XML"
- Chapter 5, "Customizing Content with XML: Dynamic News Application"

Main Tasks Involved

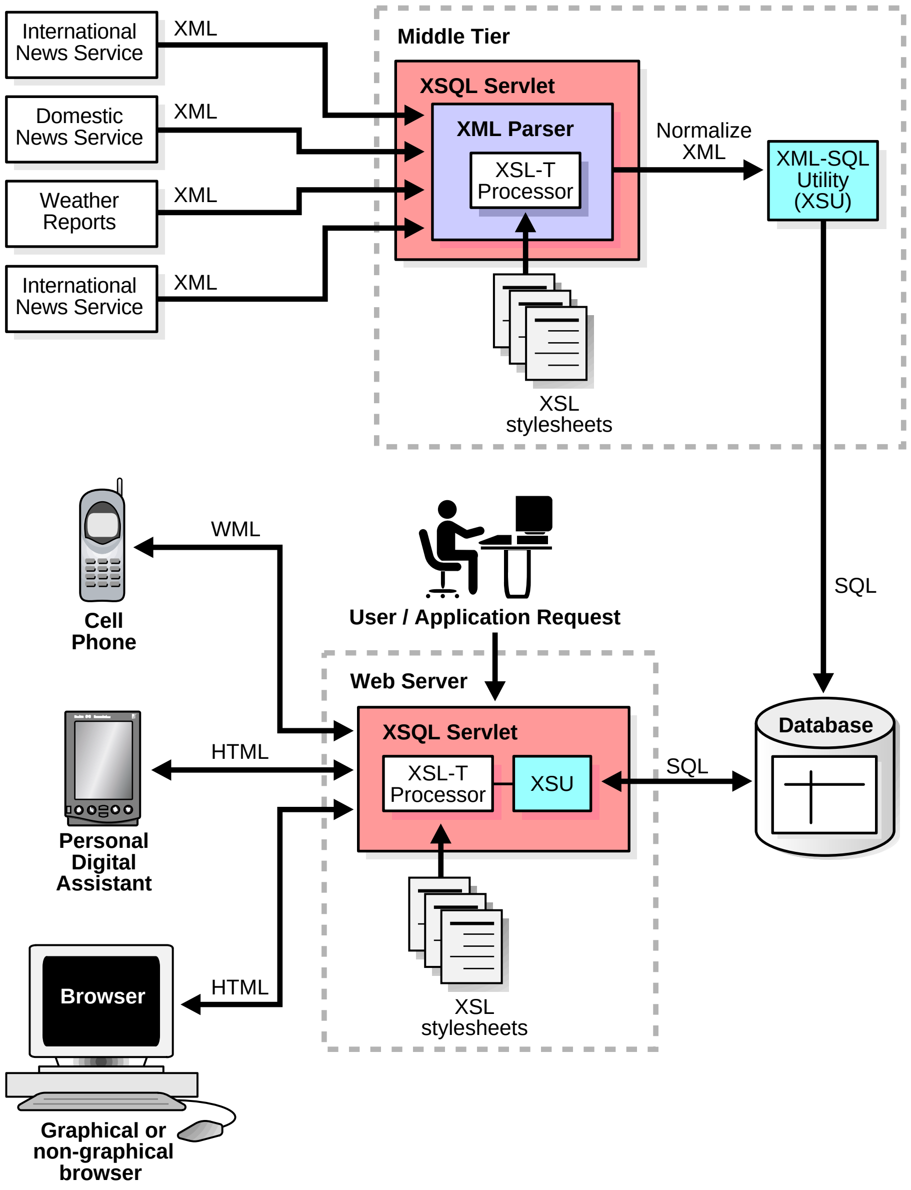
These are the main tasks involved in Scenario 2:

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

Oracle XML Components Used

- XML Parser for Java v2
- XML SQL Utility (XSU)
- XSQL Servlet

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



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

Problem

Company X needs data interactively delivered to a thin client.

Solution

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

Main Tasks Involved

See Chapter 7, "Customizing Discoverer 4i Viewer with XSL", and also the Reports9i chapter in Oracle9i Application Developer's Guide - XML.

Oracle XML Components Used

- XML Parser for Java and XSLT Processor

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

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

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

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

- Scenario 4. B2B Messaging: Online Multivendor Shopping Cart Design Using XML
- Scenario 5. B2B Messaging: Using Oracle XML Components and Advanced Queueing for an Online Inventory Applicationn
- Scenario 6. B2B Messaging: Using Oracle XML-Enabled Technology and AQ for Multi-Application Integration

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

These scenarios are illustrated with case studies in Part III, "XML Data Exchange".

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

Problem

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

Solution

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

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

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

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

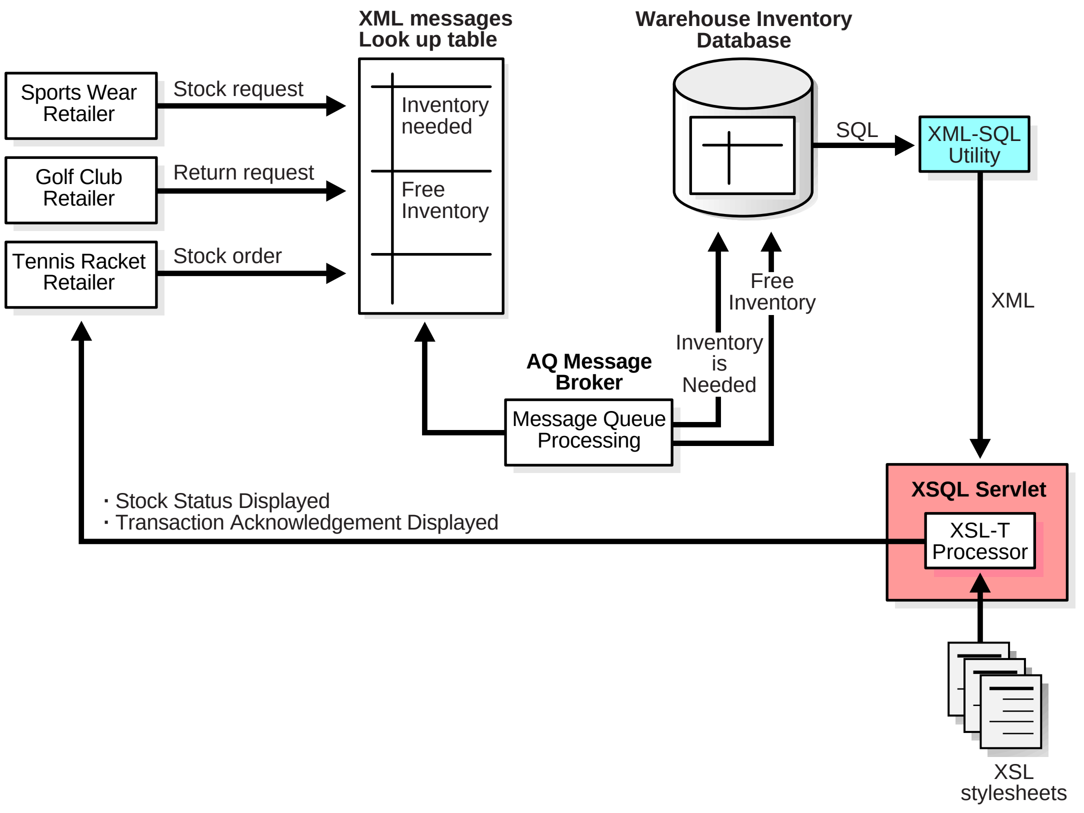
See Also: Chapter 8, "Online B2B XML Application: Step by Step", for examples of similar implementations.

- *Oracle9i Application Developer's Guide* - *XML*, the Advanced Chapter chapter (9) for some basic information about using XML with Advanced Queueing
- *Oracle9i Application Developer's Guide* - *Advanced Queuing*

Oracle XML Components Used

- Oracle XML Parser
- XML SQL Utility
- XSQL Servlet

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



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

Problem

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

Solution

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

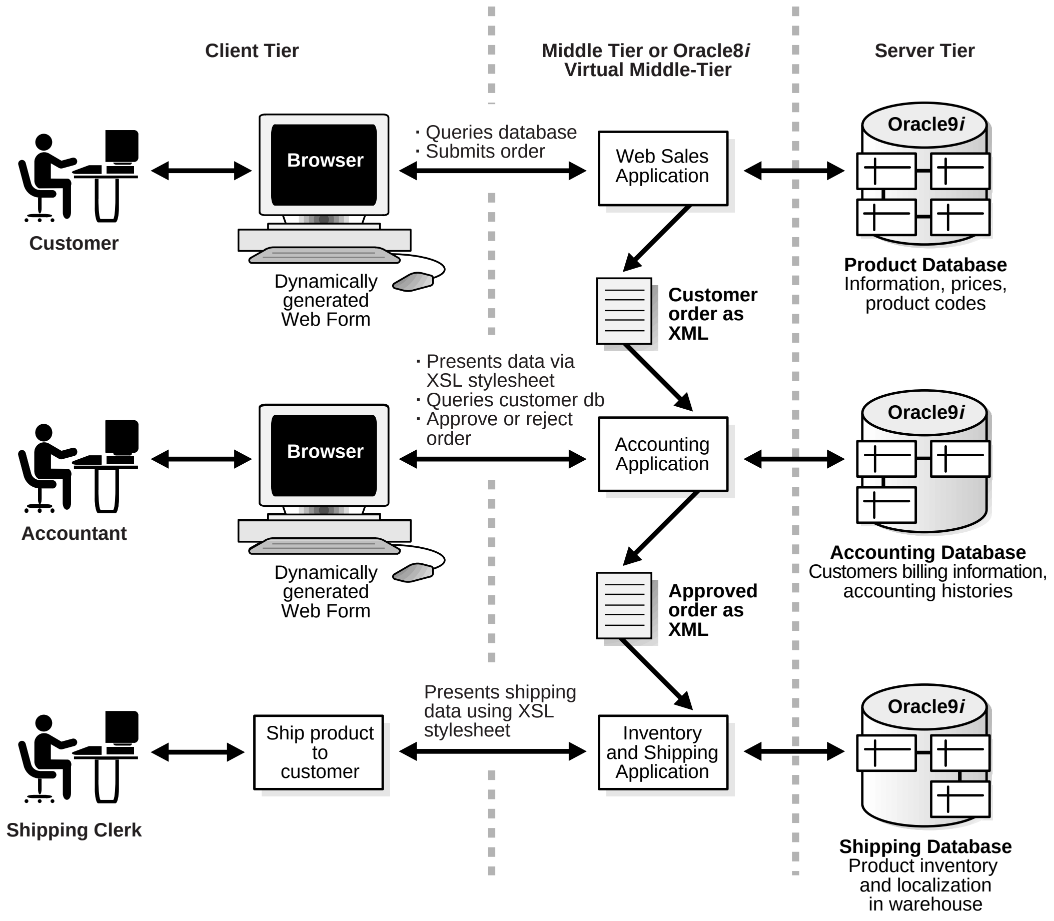
See Also: Chapter 8, "Online B2B XML Application: Step by Step", in this manual, for examples of similar implementations.

- *Oracle9i Application Developer's Guide* - *XML*, the Advanced Chapter chapter (9) for some basic information about using XML with Advanced Queueing
- *Oracle9i Application Developer's Guide* - *Advanced Queuing*

Oracle XML Components Used

- XML Parser
- XSLT Processor

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


Client Tier
Middle Tier or Oracle8*i* Virtual Middle-Tier
Server Tier
Customer
Browser
· Queries database
· Submits order
Web Sales Application
Oracle9*i*
Product Database
Information, prices, product codes
Dynamically generated Web Form
Customer order as XML
· Presents data via XSL stylesheet
· Queries customer db
· Approve or reject order
Accountant
Browser
Accounting Application
Oracle9*i*
Accounting Database
Customers billing information, accounting histories
Dynamically generated Web Form
Approved order as XML
Presents shipping data using XSL stylesheet
Shipping Clerk
Ship product to customer
Inventory and Shipping Application
Oracle9*i*
Shipping Database
Product inventory and localization in warehouse

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

Problem

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

Solution

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

Main Tasks Involved

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

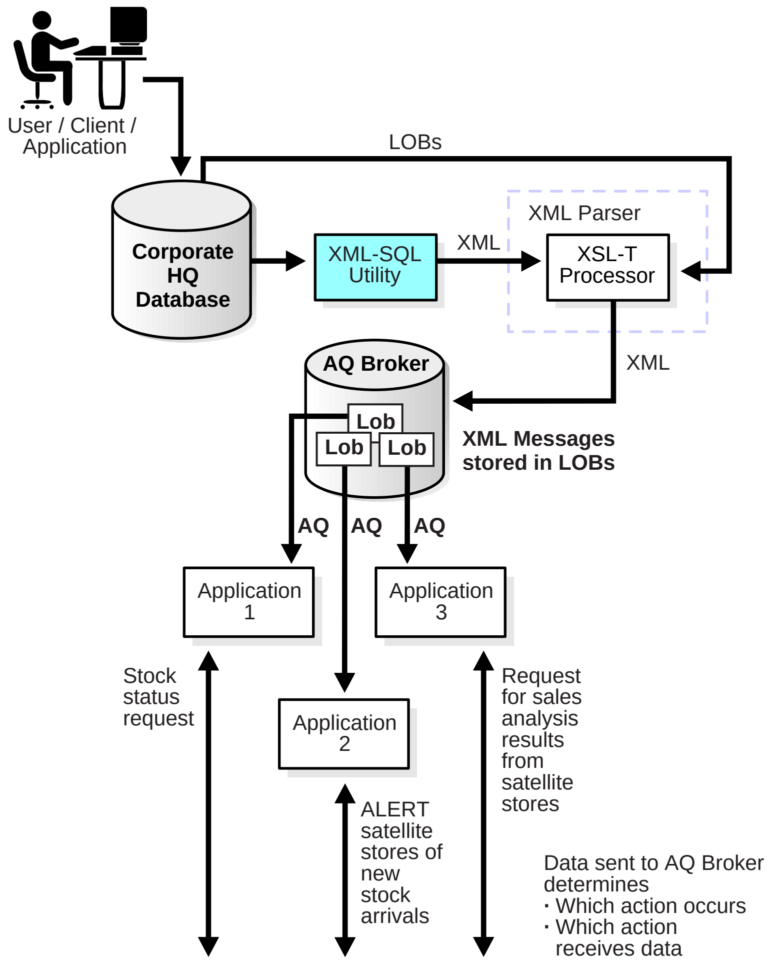
See Also: Chapter 8, "Online B2B XML Application: Step by Step", for examples of similar implementations.

- *Oracle9i Application Developer's Guide - XML*, the Advanced Chapter chapter (9) for some basic information about using XML with Advanced Queueing
- *Oracle9i Application Developer's Guide - Advanced Queuing*

Oracle XML Components Used

- XML Parser
- XSLT Processor
- XML SQL Utility (XSU)

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


User / Client / Application
LOBs
Corporate HQ Database
XML-SQL Utility
XML
XML Parser
XSL-T Processor
XML
AQ Broker
Lob
Lob
Lob
XML Messages stored in LOBs
AQ
AQ
AQ
Application 1
Application 3
Application 2
Stock status request
Request for sales analysis results from satellite stores
ALERT satellite stores of new stock arrivals
Data sent to AQ Broker determines
· Which action occurs
· Which action receives data

Part II

Managing Content and Documents with XML

Part II of this manual describes case studies that show ways to implement XML-based content and document management.

Part II contains the following chapters:

3 Oracle9*i* AS Wireless Edition and XML

This chapter contains the following sections:

Introducing Oracle9i AS Wireless Edition (Portal-to-Go)

Oracle9i Application Server Wireless Edition (Oracle9i AS Wireless Edition) allows carriers, enterprises, and Internet companies to wirelessly enable any new or existing Internet applications or content for any wireless Internet device - including Smartphones, pagers, PDAs, and so on.

OracleMobile, a division of Oracle Corp., is a wireless application service provider (WASP) with a complete service offering for wireless Web site creation and hosting based on Oracle9i Application Server Wireless Edition technology.

Oracle9i Application Server Wireless Edition

Content Adapters convert any content to XML. Transformers convert the XML to any markup language supported by any device (HTML, WML, HDML, VoiceXML, VoxML, SMS, etc.). Oracle9i AS Wireless Edition's open architecture and use of XML technology ensures support of current and emerging standards. Oracle9i AS Wireless Edition provides for location based services, wireless messaging, wireless m-commerce, and extensive personalization for users and devices.

Most Web clients are PCs, but according to the Meta Group, “By 2003, over 50% of internet access will be by non-PCs.”

Oracle9i AS Wireless Edition (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

Oracle9i AS Wireless Edition portals contain:

- Content Adapters - Transforms any content to independent XML
- Content Transformers - Transforms XML to device-specific markup language
- Service Designer - Specifies how web services are designed and managed. The services deliver data to mobile devices.
- Personalized Portal - Allows users to personalize services they access
- Request Manager - Recognizes user devices and services request

This chapter describes how Oracle9i AS Wireless Edition 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

XML Parser for Java v2 is used in Oracle9i AS Wireless Edition Adapters and Transformers. The XSLT package of the XML Parser for Java is also used.

Oracle9*i* AS Wireless Edition (Portal-To-Go) Features

The Oracle9i AS Wireless Edition features include the following support:

- Apache and Apache JServ.
- All mobile devices.
- Customization of device output, such as explicit settings of output variable names.
- Handling of single-byte, multi-byte and fixed-width encoding schemes (and special characters, such as “$”).
- Oracle9iAS Wireless includes a scalable, user-customizable notification engine that 'pushes' information to any messaging-enabled device (SMS, e-mail, and so on).
- Comprehensive support for the development of 'Location Based Services'.

- Filtering services for simple automatic filtering and translation of all existing content.
- Flexible portal and personalization features out of the box, simplifying the wireless user’s experience including location marks, bookmarks and object control.
- Advanced transaction and database logging, event management and performance monitoring.

See Also:

- *Oracle9i AS Wireless Edition Installation Guide*, for more details on repository upgrades.
- http://otn.oracle.com/products/iaswe/

What’s Needed to Run Oracle9i AS Wireless Edition

Oracle9i AS Wireless Edition requires the following:

- Oracle8*i*, Release 8.1.5 or above
- One of the following servers:
  - Oracle9*i* Application Server
- Java Configuration Requirements
  - Service Designer. The Oracle9i AS Wireless Edition Service Designer requires JDK 1.2.2. You can install JDK 1.2.2 from the Oracle9i AS Wireless Edition 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 Oracle9i AS Wireless Edition server component runs with JDK 1.1 or 1.2. JDK 1.2 has improved performance.

Oracle9i AS Wireless Edition: Supported Devices and Gateways

Transformers

Oracle9i AS Wireless Edition 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 Oracle9i AS Wireless Edition support to other devices.

WAP Gateways

Oracle9i AS Wireless Edition has been successfully tested with the following WAP gateways:

- Phone.com UP.link Gateway
- Nokia WAP Gateway
- Ericsson WAP Gateway
- Infinite Technologies WAPLite

How Oracle9i AS Wireless Edition Works

Figure 3–1 shows how Oracle9i AS Wireless Edition (Portal-to-go) works. When an end-user requests an Oracle9i AS Wireless Edition service, this is what happens:

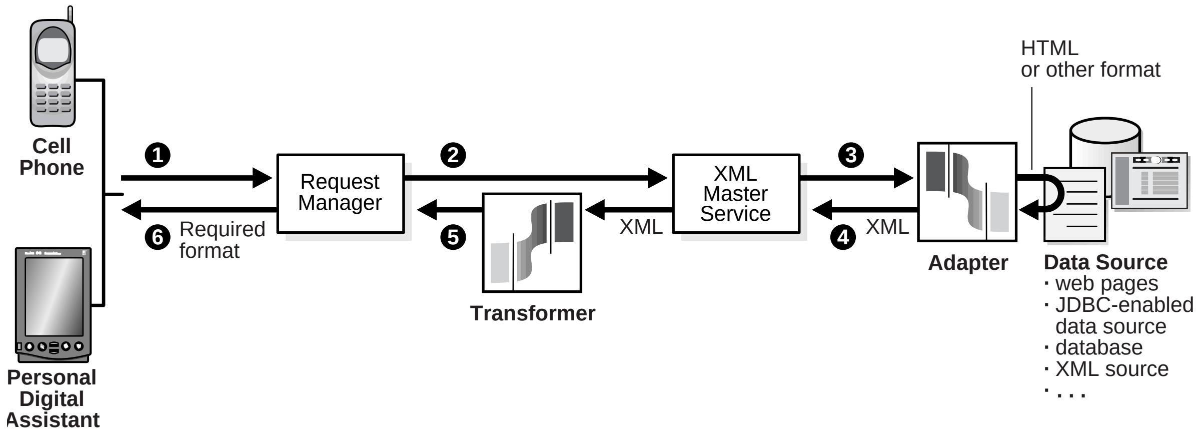
1. Oracle9i AS Wireless Edition’s Request Manager performs user-level preprocessing, including authentication.
2. Request Manager sends a request to the corresponding Master Service.
3. Master Service invokes an adapter to retrieve the requested content.
4. Adapter returns the content in XML.
5. Transformer 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 Oracle9i AS Wireless Edition’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 3–1 How Oracle9i AS Wireless Edition Works



Oracle9i AS Wireless Edition Components

Oracle9i AS Wireless Edition Services

A Oracle9i AS Wireless Edition service encapsulates a unit of information requested by, and delivered to, a Oracle9i AS Wireless Edition client. Examples of services include:

- Stock quotes
- News
- Maps
- Email

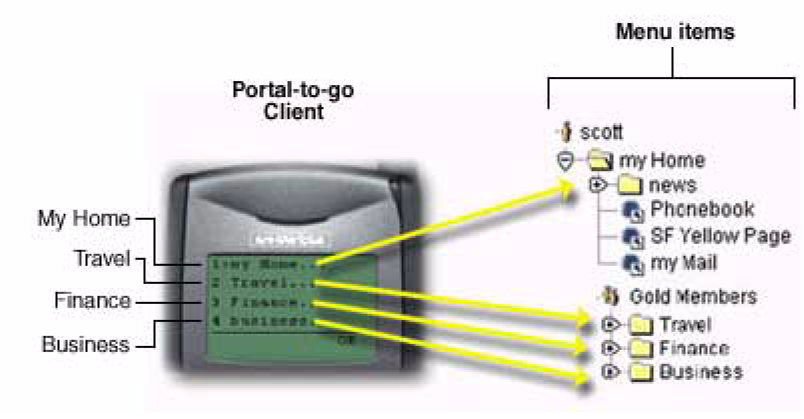
You can build services from an existing Web site, a query to any database, or any XML source.

Master Service

A Master Service is a Oracle9i AS Wireless Edition 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 3–2 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 Oracle9i AS Wireless Edition 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.

Oracle9i AS Wireless Edition Adapters

A Oracle9i AS Wireless Edition Adapter is a Java application that retrieves data from an external source and renders it in Oracle9i AS Wireless Edition XML. When invoked by a Master Service, an Adapter returns an XML document that contains the service content. Adapters provide the interface between the Oracle9i AS Wireless Edition server and the content source.

An Adapter does the following:

- Connects to a data source
- Retrieves content
- Converts the content to Oracle9i AS Wireless Edition XML

Oracle9i AS Wireless Edition 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 Oracle9i AS Wireless Edition XML. This is a well-formed, valid XML document that complies with the Oracle9i AS Wireless Edition DTD.

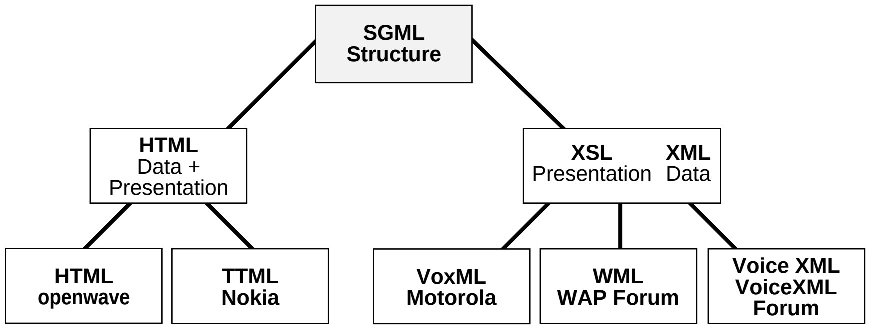
Oracle9i AS Wireless Edition Transformers

Oracle9i AS Wireless Edition Transformers are Java programs or XSL-T stylesheets that convert an XML document into the target or another Oracle9i AS Wireless Edition 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. Oracle9i AS Wireless Edition 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 designed specifically for handheld devices.
- Plain Text - Converts content for Short Message Service-capable devices and email applications.

Figure 3–3 illustrates these markup languages and their derivation.

Figure 3–3 Oracle9i AS Wireless Edition 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 Oracle9i AS Wireless Edition

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? Oracle9i AS Wireless Edition 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

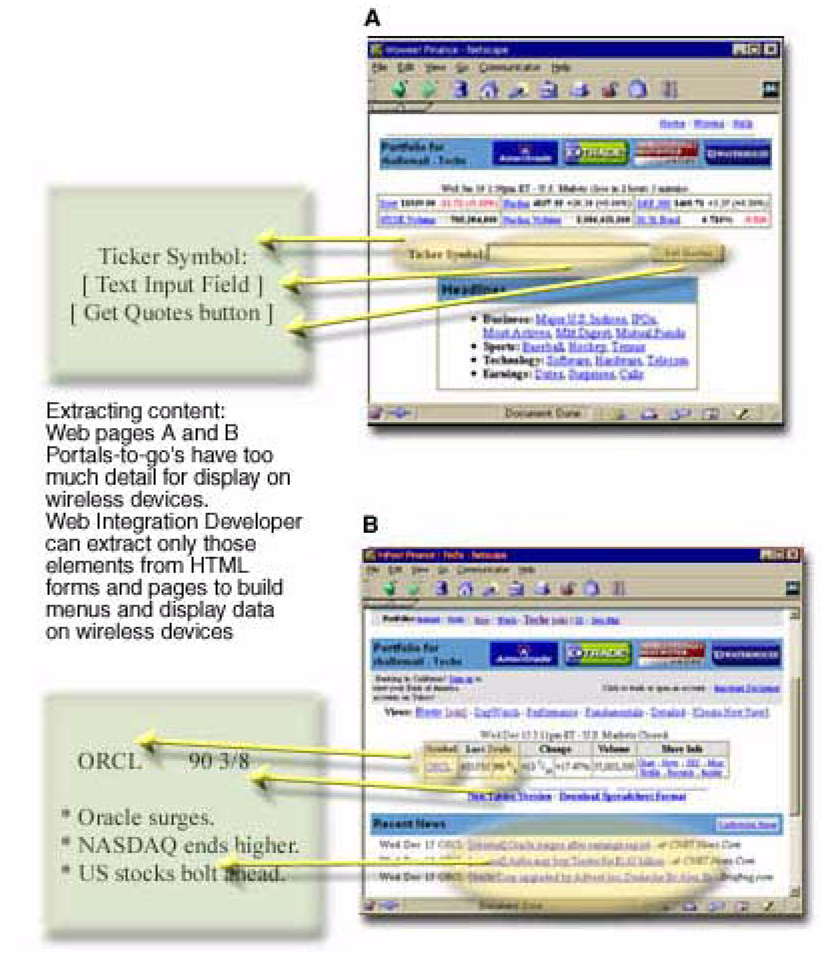
Hand-held devices cannot display as much information as a desktop monitor, so you have to be selective. Figure 3–4 shows two, deliberately undecipherable, Web pages from a Stock Data application.

- 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.

You then can use the Web Integration Adapters to convert your content to XML.

Figure 3–4 Extracting Elements from HTML Pages For Display on Wireless Devices



Converting to XML

A Oracle9i AS Wireless Edition 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

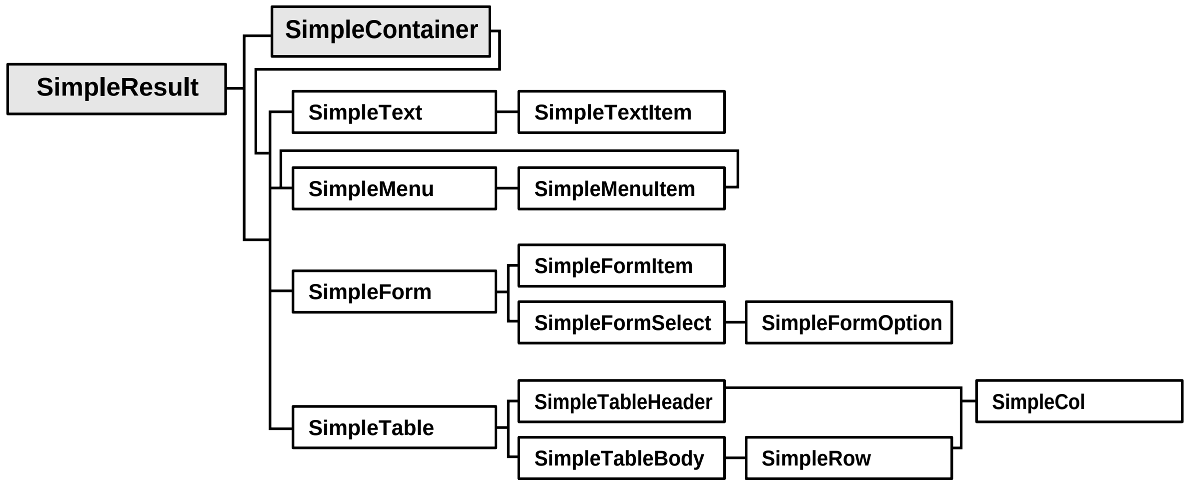
Adapter output must be XML to be generic. The key is to define an XML document type that can represent any data type you might want to display on any device. The document type is defined by a Document Type Definition (DTD). A DTD is a file that provides a grammar for a class of XML documents by describing the elements it can contain.

To create a truly universal intermediate data format, Oracle9i AS Wireless Edition 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 3–5 illustrates the Simple Result DTD content model.

Figure 3–5 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

Oracle9i AS Wireless Edition 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 3–1 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 3–1 XML for Input and Results Page Generated by StockData Adapter

XML for Input Page	XML Results Page: Quote and Headlines Page
<SimpleResult>	<SimpleResult>
<SimpleText>	<SimpleText title="Quote Results">
<SimpleTextItem name = "TickerField">	<SimpleTextItem name="Ticker">
Ticker Symbol:	ORCL
</SimpleTextItem>	</SimpleTextItem>
</SimpleText>	<SimpleTextItem name="Price">
<SimpleForm title="Input Form">	90 3/8
<SimpleFormItem name="Ticker">	</SimpleTextItem>
</SimpleFormItem>	</SimpleText>
<SimpleFormButton name="submitBtn">	<SimpleText title="Headlines">
Get Quote	<SimpleTextItem name = "Headline1">
</SimpleFormButton>	* Oracle surges.
</SimpleForm>	</SimpleTextItem>
</SimpleResult>	<SimpleTextItem name = "Headline2">
	* NASDAQ closes higher.
	</SimpleTextItem>
	<SimpleTextItem name = "Headline3">
	* US stocks bolt ahead.
	</SimpleTextItem>
	</SimpleText>
	</SimpleResult>

Sample Adapter Classes

The following code example shows how Adapters are implemented in Java. You can it to create your own Adapters for custom content sources.

"Oracle9i AS Wireless Edition Adapter Example 1: Greeting Users by Name" is a simple, but complete, Adapter implementation that greets users by name.

Oracle9i AS Wireless Edition Adapter Example 1: Greeting 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

Oracle9i AS Wireless Edition 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:

- *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. Oracle9i AS Wireless Edition uses XSL style sheets by default. See "Oracle9i AS Wireless Edition XSL Stylesheet Transformer Example 1: Converting Simple Result Documents to Plain Text".
- *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 "Oracle9i AS Wireless Edition Java Transformer Example 1: Converting Simple Result Elements to Another Format".

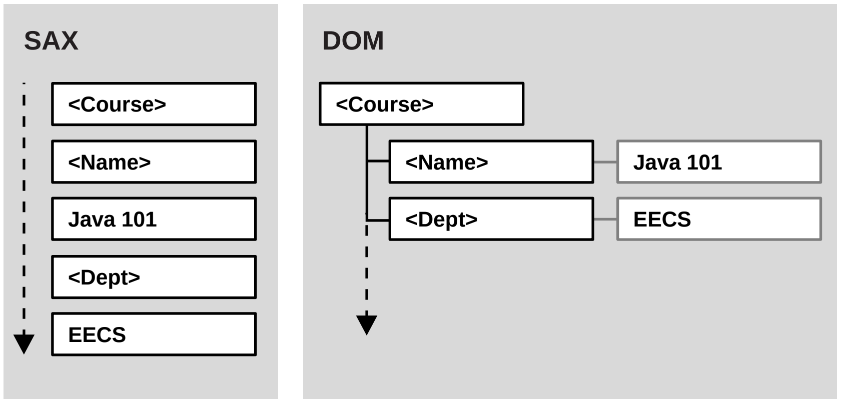
Oracle9i AS Wireless Edition: Java Transformers

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 3–6.

Figure 3–6 The DOM and SAX Interfaces



Oracle9i AS Wireless Edition 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.

Oracle9i AS Wireless Edition 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();
    }
}
```

Oracle9i AS Wireless Edition: 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.

Oracle9i AS Wireless Edition XSL Stylesheet Transformer Example 1: Converting Simple Result Documents to Plain Text

This XSL Transformer example is included in the Oracle9i AS Wireless Edition 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 3–2 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 3–2 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>`
	`   <p>Quote</p>`
	`      <p>Ticker: ORCL</p>`
	`      <p>Price: 90 3/8</p>`
	`    <p>Headlines:</p>`
	`     <p>* Oracle surges.</p>`
	`     <p>* NASDAQ closes higher.</p>`
	`     <p>* US stocks bolt ahead.</p>`
	`</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>`

Oracle9i AS Wireless Edition 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>
```

Oracle9i AS Wireless Edition Stylesheet TransformerExample 3: XSL Java Extension

The SimpleResult XML has been extended by adding a new element called SimpleDB. This element is used to execute your INSERT, UPDATE, and DELETE statements on any database or your PLSQL on an Oracle database.

For example, you can use this feature for an advanced billing system, where the cost is defined by the content provider. You can also use all the database standard features (UTL_SMTP, UTL_FILE...) to extend the master services capabilities.

To do this, a new Java class has been created called, `oracle.panama.core.xform.MyXslExtension`, with a method called `processDB()`.

To test this feature, copy `MyXslExtension.class` to

%ORACLE_HOME%/panama/server/classes/oracle/panama/core/xform and add onto your device transformer the following header:

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

xmlns:p2g="http://www.oracle.com/XSL/Transform/java/oracle.panama.core.xform.XSL
Java" exclude-result-prefixes="p2g"

xmlns:myxsl="http://www.oracle.com/XSL/Transform/java/oracle.panama.core.xform.M
yXslExtension">
```

Then add the following statement to your transformer body:

```
<!-- Execute SimpleDB-->
<xsl:template match="SimpleDB">
      <xsl:value-of select="p2g:processDB(.)"/>
</xsl:template>
```

or

```
<!-- Bypass SimpleDB-->
<xsl:template match="SimpleDB">
      <xsl:apply-template/>
</xsl:template>
<xsl:template match="SimpleDBItem"/>
```

Here's an example of SimpleResult, generated by a Result Transformer, with the SimpleDB element:

```
<SimpleResult>
<SimpleContainer>
  <SimpleText>
     <SimpleTextItem>Symbol : ORCL</SimpleTextItem>
     <SimpleTextItem>Value (USD): 76 15/32</SimpleTextItem>
     <SimpleTextItem>Value (HKD): 592.59</SimpleTextItem>
  </SimpleText>
  <SimpleDB jdbc="jdbc:oracle:thin:@hkpsnt3.hk.oracle.com:1521:orcl"
user="scott" password="tiger">
    <SimpleDBItem type="SQL">
       insert into quote values ('orcl','ORCL','76 15/32',null, null)
    </SimpleDBItem>
    <SimpleDBItem type="PLSQL">
      begin
       insert into quote values ('orcl','ORCL','76 15/32',null,sysdate);
      end;
    </SimpleDBItem>
  </SimpleDB>
</SimpleContainer>
</SimpleResult>
```

This is just an example and of course your can add your own extension by creating other Java methods.

MyXslExtension.java

```
// Allen M.K. YAP - Sales Consultant
// Oracle System Hong Kong Ltd.
// Email : Manh-Kiet.Yap@oracle.com

// File : MyXslExtension.java
// Date : 28/09/2000

package oracle.panama.core.xform;

import oracle.panama.core.xml.XML;
import oracle.panama.core.admin.L;
import org.w3c.dom.*;
import java.sql.*;

public class MyXslExtension
{
    private static String getTextValue(Element element)
    {
        Node node = element.getFirstChild();
        if(node != null && node.getNodeType() == 3)
            return node.getNodeValue();
        else
            return "";
    }

    public static String processDB (NodeList nlist) {

        Element element  = (Element) nlist.item(0);

        String out = new String();
        boolean fail = false;
        String jdbc_conn = element.getAttribute("jdbc");
        String jdbc_user = element.getAttribute("user");
        String jdbc_pass = element.getAttribute("password");
        try {
          Connection conn = DriverManager.getConnection (jdbc_conn,jdbc_
user,jdbc_pass );

          NodeList nodelist = element.getElementsByTagName("SimpleDBItem");
          for(int i = 0; i < nodelist.getLength(); i++)
          {
            Element st = (Element)nodelist.item(i);

            if ( (element.getAttribute("type")).equals("SQL")) {//SQL
              try {
                PreparedStatement pstmt=null;
                pstmt = conn.prepareStatement(getTextValue(st));
                pstmt.executeUpdate();
                pstmt.close();
              }
              catch (SQLException e) {
               L.e("SQL fails : "+e);
                out.concat(getTextValue(st)+"\nSQL fails :\n "+e+"\n");
                fail=true;
              }
            }
            else {//PLSQL
              try {
                CallableStatement plstmt = conn.prepareCall (getTextValue(st));
              plstmt.execute();
              plstmt.close();
            }
            catch (SQLException e) {
                fail=true;
               L.e("PLSQL fails : "+e);
                out.concat(getTextValue(st)+"\nPLSQL fails :\n "+e+"\n");
            }
           }
        }

        conn.commit();
        conn.close();
        if (fail) return (out);
        else return ("DB actions successfully completed.");
      }
      catch (SQLException e) {
        L.e(e);
        return ("DB actions failed");
      }

    }

    public MyXslExtension()
    {
    }
}
```

Oracle9i AS Wireless Edition Case Study 1: Extending Online Drugstore's Reach

An online drugstore is using Oracle® Oracle9i AS Wireless Edition 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 Oracle9i AS Wireless Edition extends the existing Internet site to hand-held wireless devices. In this case Oracle9i AS Wireless Edition 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.

Oracle9i AS Wireless Edition 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.

Oracle9i AS Wireless Edition 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® Oracle9i AS Wireless Edition.

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.

Oracle9i AS Wireless Edition Case Study 3:Online Auction Sites

Online auction sites can extend their accessibility and usability to their customers by offering them an option for shopping from cell phone, PDAs, or other mobile devices.

4

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 Oracle9*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://otn.oracle.com/sample_code/index.htm
- http://otn.oracle.com/tech/xml/xsql_servlet/index.htm then select Sample Code

What's Needed to Run XML Flight Finder

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 Oracle9*i* Application 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 Oracle9*i* Application Server.

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 Oracle9*i*, Flight Finder uses the following products and technologies:

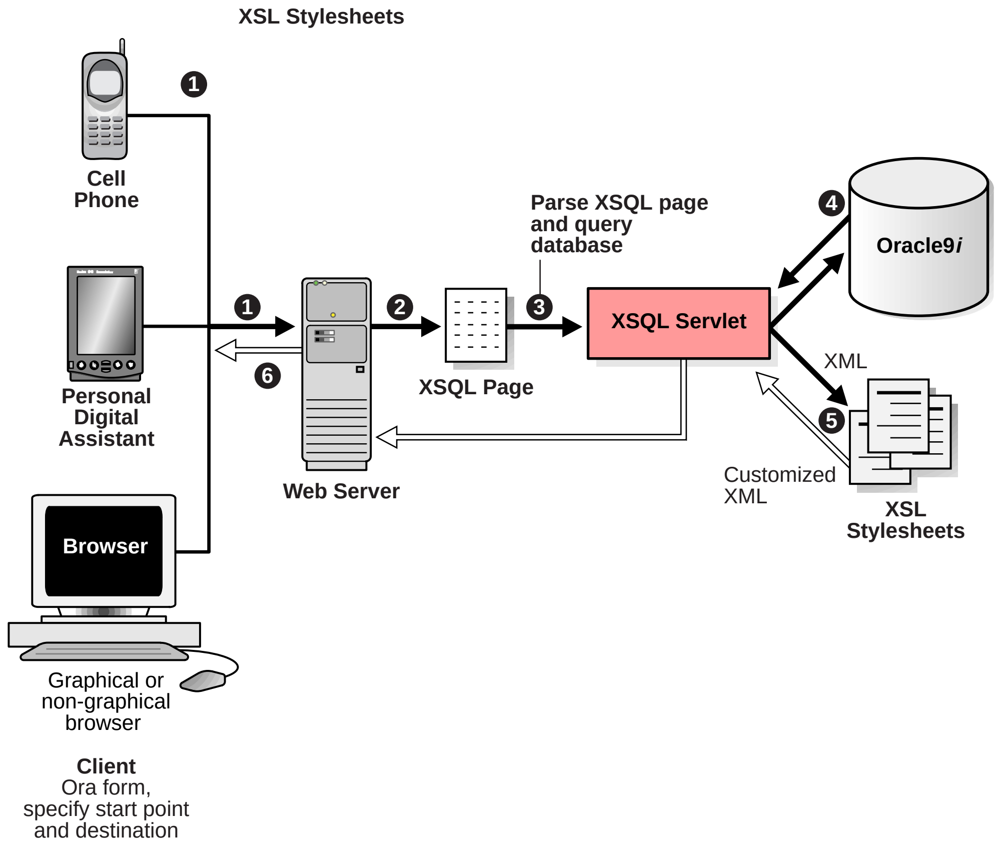
- SQL, the standard for accessing business data
- Oracle XSQL Servlet, which processes queries defined in XSQL pages. XSQL pages are XML documents that contain SQL code. XSQL Servlet outputs the result set as XML.
- XSLT, which defines an open standard for transforming XML for target devices.

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 4–1 shows how Flight Finder works.

Figure 4–1 XML Flight Finder


XSL Stylesheets
1
Cell Phone
Parse XSQL page and query database
4
Oracle9*i*
1
2
3
XSQL Servlet
6
XSQL Page
XML
5
Personal Digital Assistant
Web Server
Customized XML
XSL Stylesheets
Browser
Graphical or non-graphical browser
Client
Ora form, specify start point and destination

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 Oracle9i, 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 Oracle9i Lite to store and retrieve XML data. You can also run these components on a middle tier such as Oracle9i Application Server, 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.

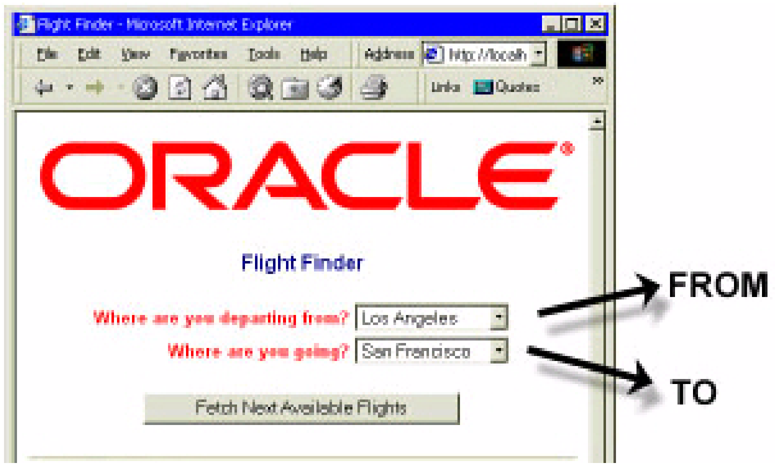
`flightFinderResult` tag defines a structure that assigns values to parameters in a query. The tag also identifies a namespace for defining the xsql keyword and tells the XSQL servlet to use the (predefined) database connection named fly.

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 4–2 shows the Flight Finder browser form and how it is used to enter FROM information (Los Angeles) and TO information (San Francisco).

Figure 4–2 Using XSQL Servlet to Process Queries and Output Result as XML: Entering FROM and TO on the Flight Finder Browser Form


Flight Finder - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localh
Links Quotes
ORACLE
Flight Finder
Where are you departing from? Los Angeles
Where are you going? San Francisco
Fetch Next Available Flights
FROM
TO

```
<?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://otn.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 foregoing XML document contains nodes named ROWSET, ROW, CODE, and so on. 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>
```

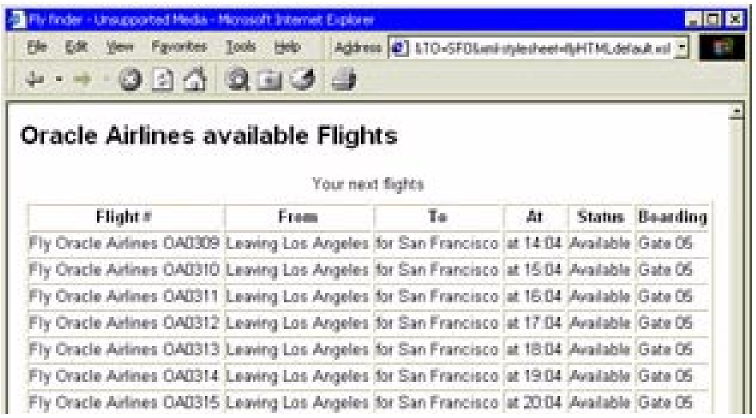
In this example, the formatting is simple: it just prepends a string to the contents of each node. For example, when the XSLT processor gets to the CODE node, it prepends the string "Fly Oracle Airlines " to the value of that node. The resulting HTML looks like this:

```
<!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 4–3 shows the results displayed on the browser after the stylesheet has been applied to the XML.

Figure 4–3 Flight Finder: Results After Formatting the XML with Stylesheets


Fly finder - Unsupported Media - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address
&TO=SFO&xml-stylesheet=flyHTMLdefault.xsl

Oracle Airlines available Flights

Your next flights

Flight #	From	To	At	Status	Boarding
Fly Oracle Airlines OA0309	Leaving Los Angeles	for San Francisco	at 14:04	Available	Gate 05
Fly Oracle Airlines OA0310	Leaving Los Angeles	for San Francisco	at 15:04	Available	Gate 05
Fly Oracle Airlines OA0311	Leaving Los Angeles	for San Francisco	at 16:04	Available	Gate 05
Fly Oracle Airlines OA0312	Leaving Los Angeles	for San Francisco	at 17:04	Available	Gate 05
Fly Oracle Airlines OA0313	Leaving Los Angeles	for San Francisco	at 18:04	Available	Gate 05
Fly Oracle Airlines OA0314	Leaving Los Angeles	for San Francisco	at 19:04	Available	Gate 05
Fly Oracle Airlines OA0315	Leaving Los Angeles	for San Francisco	at 20:04	Available	Gate 05

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

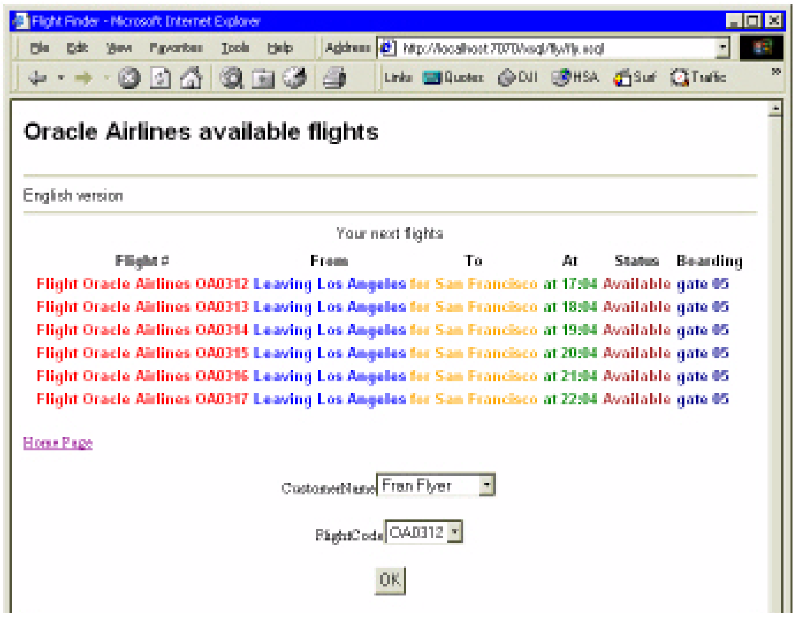
This section describes how the Flight Finder takes input from a user, converts it to XML, then writes it to the database.

1 Taking the User's Input

The first step is getting user input.

Figure 4–4 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 4–4 Flight Finder: HTML Form Displaying Results of a Query About Flights From Los Angeles, to San Francisco


Flight Finder - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:7070/xsql/fly/fly.xsql
Links Quotes DJI MSA Surf Traffic

Oracle Airlines available flights

English version

Your next flights

Flight #	From	To	At	Status	Boarding
Flight Oracle Airlines OA0312	Leaving Los Angeles	for San Francisco	at 17:04	Available	gate #5
Flight Oracle Airlines OA0313	Leaving Los Angeles	for San Francisco	at 18:04	Available	gate #5
Flight Oracle Airlines OA0314	Leaving Los Angeles	for San Francisco	at 19:04	Available	gate #5
Flight Oracle Airlines OA0315	Leaving Los Angeles	for San Francisco	at 20:04	Available	gate #5
Flight Oracle Airlines OA0316	Leaving Los Angeles	for San Francisco	at 21:04	Available	gate #5
Flight Oracle Airlines OA0317	Leaving Los Angeles	for San Francisco	at 22:04	Available	gate #5

Home Page

CustomerName Fran Flyer

FlightCode OA0312

OK

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`.

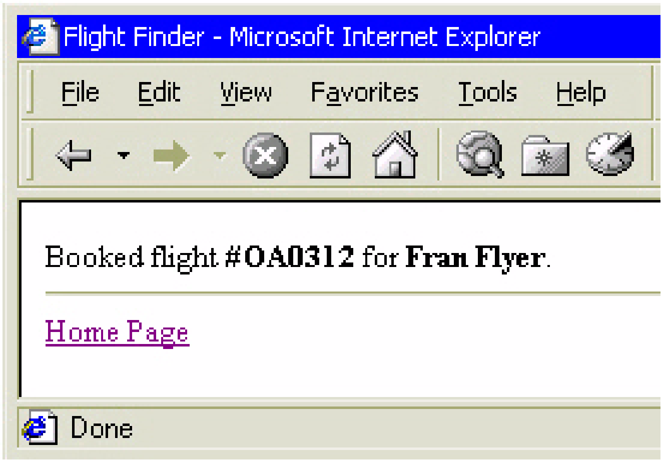
It stores the user's choices in parameters named CustomerName and FlightCode, and defines parameters named cust and code for passing the values to XSLT stylesheets. It also uses the `<xsql:dml>` tag to define a SQL statement that inserts a row into the CUSTOMERS table.

```
<?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 4–5 Flight Finder: Notifying User that Flight Was Booked



Booked flight #**OA0312** for **Fran Flyer**.

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>
```

Using Oracle9i Application Server Wireless Edition (Portal-to-Go)

Instead of writing XSQL and XSL code yourself, you can use Oracle9i Application Server (AS) Wireless Edition (Oracle Portal-to-Go).

A component of the Oracle Internet Platform, Oracle9i AS Wireless Edition 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.

Oracle9i AS Wireless Edition uses XML to isolate content acquisition from content delivery.

A Oracle9i AS Wireless Edition 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 Oracle9i AS Wireless Edition page at http://otn.oracle.com/products/iaswe.

See Also: Chapter 3, "Oracle9i AS Wireless Edition and XML".

5 Customizing Content with XML: Dynamic News Application

This chapter contains the following sections:

Introduction to the Dynamic News Application

The Dynamic News application uses Oracle XML platform components together with the Oracle9*i* database to build a web-based news service. It combines Java, XML, XSL, HTML, and Oracle9*i*.

- 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 requests.

Solution: The solution uses Oracle XML Components, Oracle9*i* 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 5–1 describes these usage choices.

Table 5–1 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 5–2 lists the servlets used in the Dynamic News application. These servlets provide entry points to the application logic:

Table 5–2 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

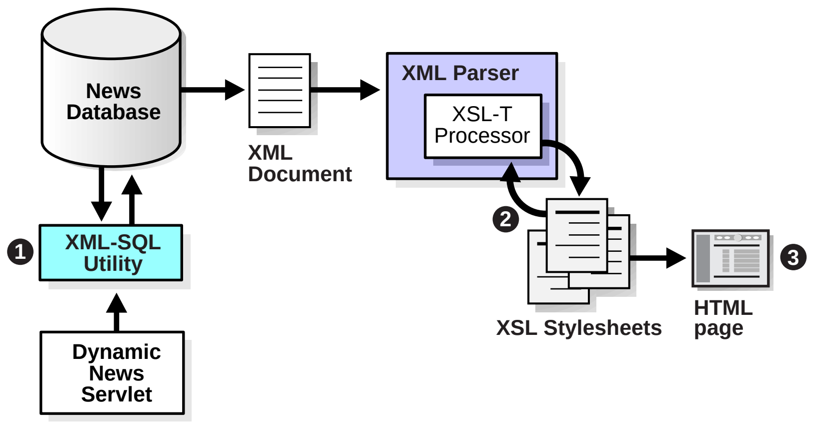
Dynamic News generates XML documents to build HTML pages:

- 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 5–1 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 5–1 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.

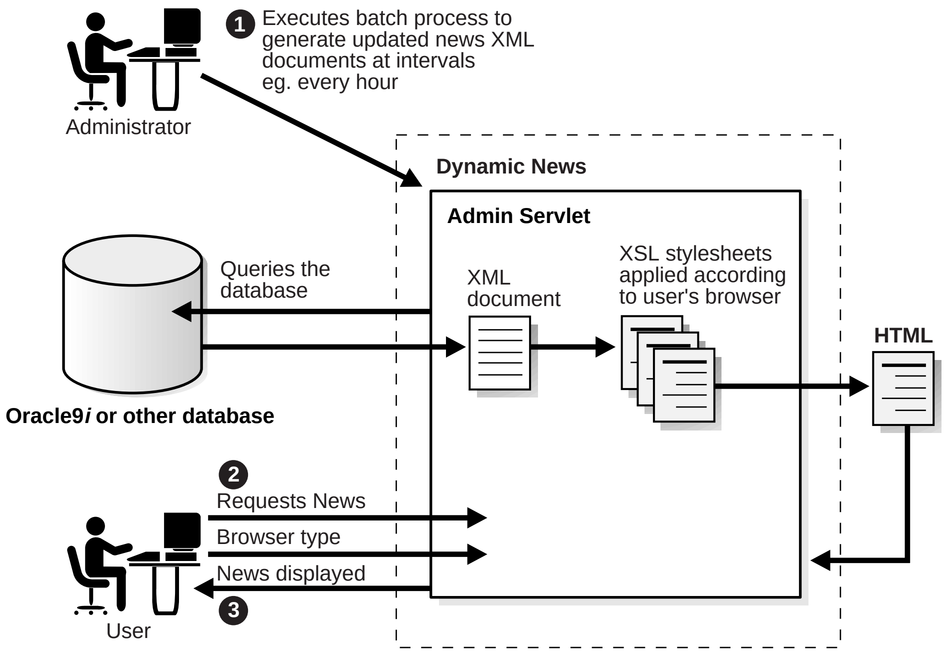
How Static Pages Works

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 5–2.

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 5–2 Dynamic News: Static Pages - Generating XML Documents


1 Executes batch process to generate updated news XML documents at intervals eg. every hour
Administrator
Dynamic News
Admin Servlet
Queries the database
XML document
XSL stylesheets applied according to user's browser
HTML
Oracle9*i* or other database
2 Requests News
Browser type
News displayed
3
User

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

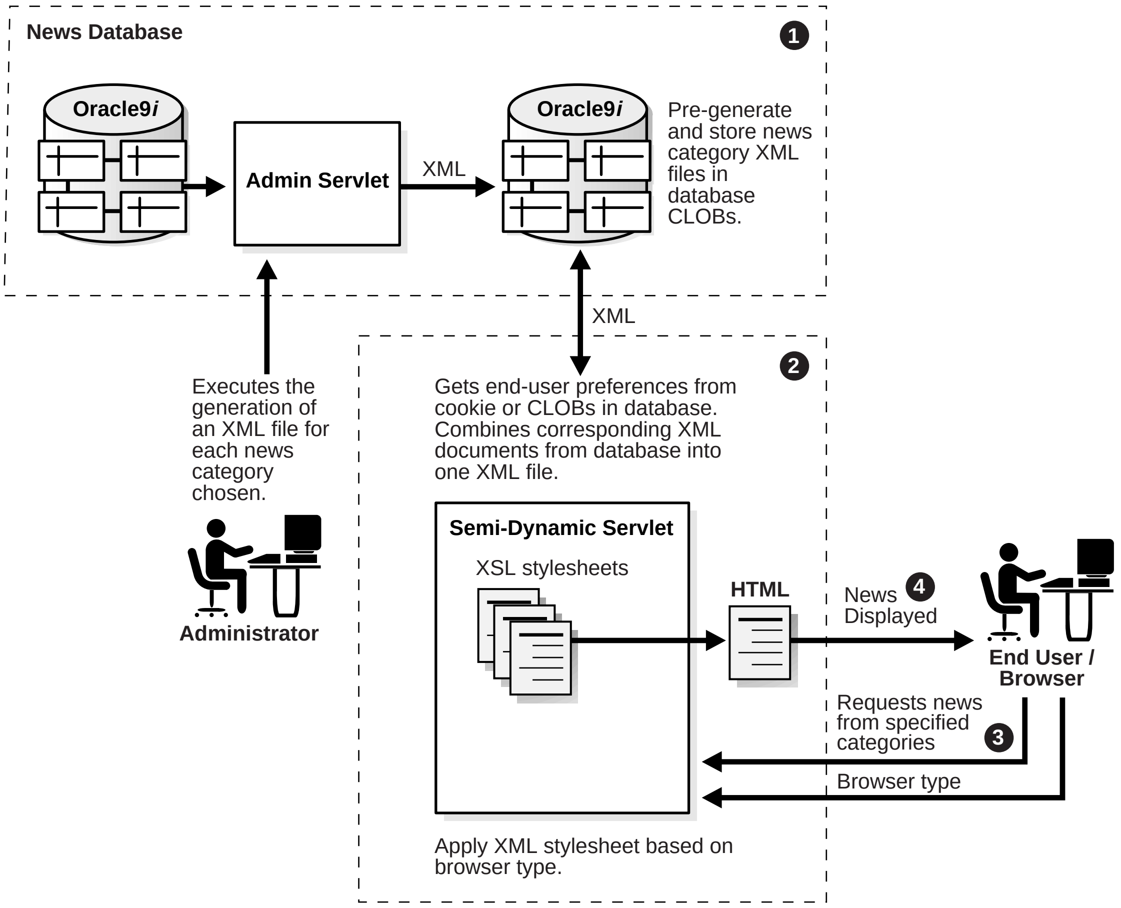
The semi-dynamic approach is useful when the data doesn't change very often and you want to give the end-user a relatively small number of choices. An application that offers more choices has to pregenerate more documents, and benefits degrade proportionally.

How Semi-Dynamic Pages Work

Figure 5–3 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 5–3 Dynamic News: Semi-Dynamic Pages - Generating XML Documents


News Database
1
Oracle9*i*
Admin Servlet
XML
Oracle9*i*
Pre-generate and store news category XML files in database CLOBs.
XML
2
Executes the generation of an XML file for each news category chosen.
Administrator
Gets end-user preferences from cookie or CLOBs in database. Combines corresponding XML documents from database into one XML file.
Semi-Dynamic Servlet
XSL stylesheets
HTML
News Displayed
4
End User / Browser
Requests news from specified categories
3
Browser type
Apply XML stylesheet based on browser type.

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

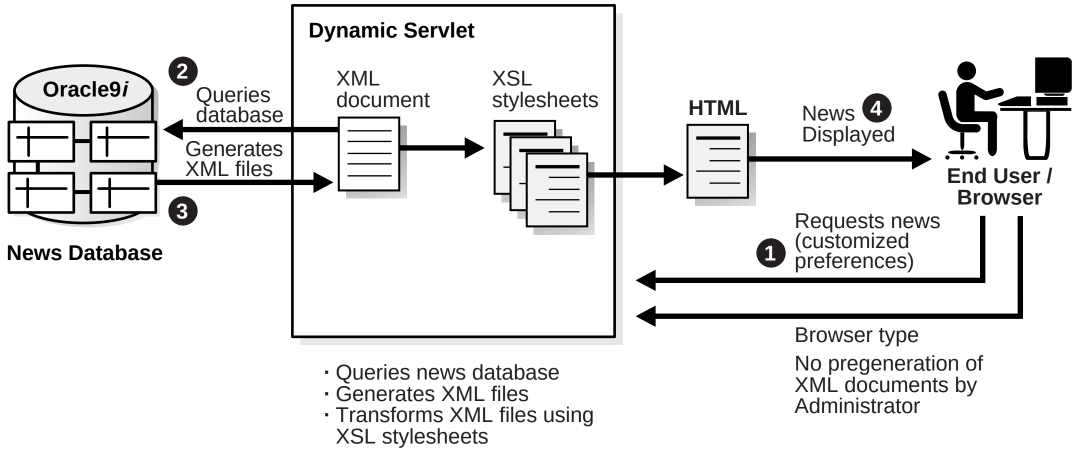
Dynamic pages are useful for delivering up-to-the-minute information, such as breaking news. They are also useful for delivering historical data, such as the closing price of any specified stock on any day in the last 10 years. It would be impractical to pregenerate documents for every possible request, but straightforward and efficient to pull the figures from the database.

How Dynamic Pages Works

Figure 5–4 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 5–4 Dynamic News: Dynamic Pages - Generating XML Documents



Personalizing Content

Oracle9*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 application 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.

6 Using Oracle9i Internet File System (9iFS) to Build XML Applications

This chapter contains the following sections:

Introducing Oracle9i Internet File System (9iFS)

Oracle9i Internet File System (9*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 Oracle9*i*, and display the document from a web server, such as Oracle9i Application Server.

Working with XML in 9iFS

*i*FS is a simple way for developer's to work with XML, where 9iFS serves as the repository for XML. 9iFS can perform the following tasks on XML documents:

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

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, 9*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 9iFS, 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.

9*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 9*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 *9iFS* supported protocols.

See Also:

- http://otn.oracle.com/products/ifs/
- *Oracle9i Internet File System Developer's Guide*.

Using the 9i*FS* Parsers

When you drop a new XML file, whose structure 9iFS understands, into a folder, Oracle XML Parser for Java can dissect the XML file and store the separate attributes in the 9iFS schema. By defining an 9iFS subclass, you are specifying which attributes go into which columns. You can also let 9iFS 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 *9iFS*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 9iFS" on page 6-7.

Standard 9iFS Parsers and Custom Parsers

In 9iFS your XML application will require a custom parser if the document format produced by the application needs it.

When Must 9iFS Parser be Explicitly Invoked?

9iFS parser is invoked depending on how the documents produced by your application are entered into 9iFS.

- If the XML documents are uploaded using protocol servers, *9iFS* 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 *9iFS* 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 *9iFS* Java API.

  The custom parser will create *9iFS* 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 *9iFS* requires a parser.

  - If the Memo documents are in XML, you can use the *9iFS* `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 *9iFS* Standard Parsers

9iFS offers several standard parsers for creating applications in *9iFS*. Table 6–1 lists the *9iFS* standard parser classes.

Table 6–1 9iFS Standard Parser Classes

Class	Description
SimpleXmlParser	Creates an object in the *9iFS* repository from an XML document body. Used as the default parser for all XML documents stored in *9iFS*. 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 *9iFS*:

- *SimpleXmlParser* for minimal customization. It works for FTP, SMB, the Windows interface, and the *9iFS* 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 *9iFS* 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 *9iFS* parser or create a custom class from scratch, implementing the `oracle.9iFS.beans.parsers.Parser` interface.

How *9iFS* XML Parsing Works

When you place an XML representation of a document in *9iFS*, `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 *9iFS* Windows interface.

1. Assume next that you have dragged an instance of the XML document, such as gking.xml, into *9iFS* folder, /iFS/system/vcards.
2. SMB performs a parser lookup based on the file extension, .xml. SMB is a protocol that lets you access *9iFS* through Microsoft Windows 95, Windows 98, and Windows NT clients. You can drag files into and out of Oracle *9iFS*, or they can edit them directly in *9iFS*.
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 *9iFS*, 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://otn.oracle.com/products/ifs

For more information about parsers, see the following classes in the *9iFS* Javadoc:

```
oracle.ifs.beans.parsers.SimpleXmlParser
oracle.ifs.beans.parsers.XmlParser
oracle.ifs.beans.parsers.SimpleTextParser
```

Rendering XML in *9iFS*

By default, the *9iFS* XML renderer reconstructs the original file, including the XML attribute tags. When you double-click the file, *9iFS* 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 *9iFS* renderer. The parsed contents of an XML file are waiting in *9iFS* 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 *9iFS* displays those files.

See Also: http://otn.oracle.com/products/iFS. Select “The XSL Custom Renderer Sample Application” then select “technical brief”, for a detailed application how to write an *9iFS* Custom Renderer.

XML and Business Intelligence

Since parsed XML files store each attribute's data in a separate, identifiable column in the *9iFS* schema, Oracle *9iFS* 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 *9iFS* file parsing files and feeding the parsed file contents to data mining tools.

Configuring *9iFS* with XML Files

To configure *9iFS*, you can write XML to customize the file system. For example, a feature of Oracle *9iFS* 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 *9iFS* subclass. Creating this subclass is as simple as:

1. Writing an XML file that follows the *9iFS* syntax for defining subclasses
2. Dragging and dropping this XML file into any folder in *9iFS*

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 *9iFS* instance.

Part III

XML Data Exchange

Part III of this manual describes case studies that show ways to implement XML data exchangest.

Part III contains the following chapters:

Oracle9i Application Developer's Guide - XML also describes other examples, applications, and FAQs.

7

Customizing Discoverer 4i Viewer with XSL

This chapter contains the following sections:

Discoverer4i 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 4i Viewer?

Oracle Discoverer 4i 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 4i 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™ 4i Viewer

This chapter provides customization examples and describes strategies for using Discoverer 4i Viewer.

- XML and XSL: Discoverer4i 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.

Discoverer4i Viewer can be driven and accessed by middle-tier B2B applications.

More Information on Discoverer 4i Viewer

For more information on Discoverer4i Viewer see:

http://technet.oracle.com/docs/products/discoverer/doc_index.htm

Discoverer 4i Viewer: Features

Discoverer 4i 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, and so on.
    - Pivot data from one axis to another to arrange it for efficient analysis and comparison
- *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 4i Viewer uses no Java and no frames, enabling even low specification browsers to be used. Where JavaScript is enabled, Discoverer uses it to 'enhance' the user interface. But JavaScript is not required and the user interface degrades gracefully if JavaScript is not enabled.
- 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 4i 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 4i Viewer: Architecture

The Discoverer 4i Viewer architecture is shown in Figure 7–1.

Discoverer 4i 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 XSLT Processor
- Discoverer 4i Viewer Servlet
- Discoverer Server interface, a Java module
- Oracle9*i* database

How Discoverer 4i Viewer Works

See Figure 7–1 to understand how Discoverer 4i Viewer works:

1. Discoverer 4i Viewer is invoked via a URL from a standard Web Browser, just like any other Web Site. The URL is processed by the 4i 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 Plus:

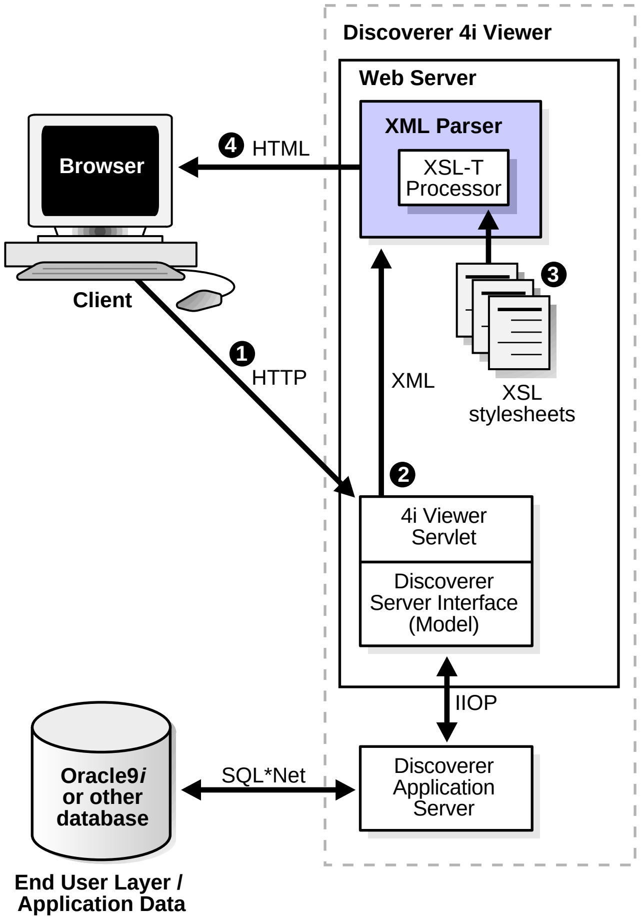
   Discoverer Server Interface. This is an applet but here it is running on the Web Server, rather than in the client's JVM as in Discoverer Plus.   The 4i Viewer Servlet communicates with Discoverer Application Server using Corba IIOP protocol.

   Discoverer 4i Viewer andDiscoverer Plus use the same Discoverer Application Server.

2. Discoverer 4i 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 (XSLT Processor).
3. This combines the XML with an XSL configuration file that defines the representation of the User Interface and,
4. XSLT Processor generates HTML to send back to the browser.

   It is the XSL file that allows the User Interface of Discoverer 4i Viewer to be customized for individual sites.

Figure 7–1 Discoverer 4i Viewer Architecture



Replicating Discoverer Application Server

The Web Server and the Discoverer 4i 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 4i Viewer for Customized Web Applications

Discoverer 4i 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 the 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/disco4iv?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 4i 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: There 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: XSLT 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 4i 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: XSLT Processor Generates HTML

The XSL and XML .

Using the XML generated in Step 2 and the standard Discoverer 4i Viewer XSL configuration file (stylesheet), these are combined in the XSLT 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 4i Viewer, the generated HTML does not use frames and therefore makes minimal demands on the browser or internet device. Hence it is easy to integrate with other web applications or portals. Where JavaScript is enabled, Discoverer uses it to 'enhance' the user interface. But JavaScript is not required and the user interface degrades gracefully if JavaScript is not enabled.

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 4i Viewer. In fact, using XSL allows a complete customized application to be made, as the next example shows.

Discoverer 4i 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 4i 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

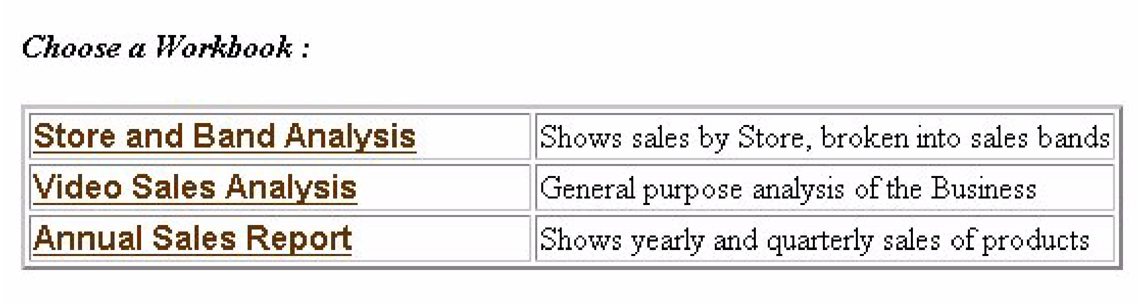
XSL file, "example1.xsl", looks like this :

```
<?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 7–2 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 7–2 List of Workbooks Viewed on Browser, XML+ example1.xsl=HTML — Before Modification



Choose a Workbook :

Store and Band Analysis	Shows sales by Store, broken into sales bands
Video Sales Analysis	General purpose analysis of the Business
Annual Sales Report	Shows yearly and quarterly sales of products

Table 7–1 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 7–1 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
<b><i>Choose a Workbook :</i></b>	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 7–1 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">`
`<font face="sans-serif">`
`<xsl:value-of select="@name"/>`
`</font>`
`</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 4i 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 7–4 for a more typical rendition.
- In Discoverer 4i Viewer, the XML and XSL are combined in the XSLT Processor on the middle tier, and not in the Web Browser.

Step 4: Customizing the XSL Stylesheet (example2.xsl)

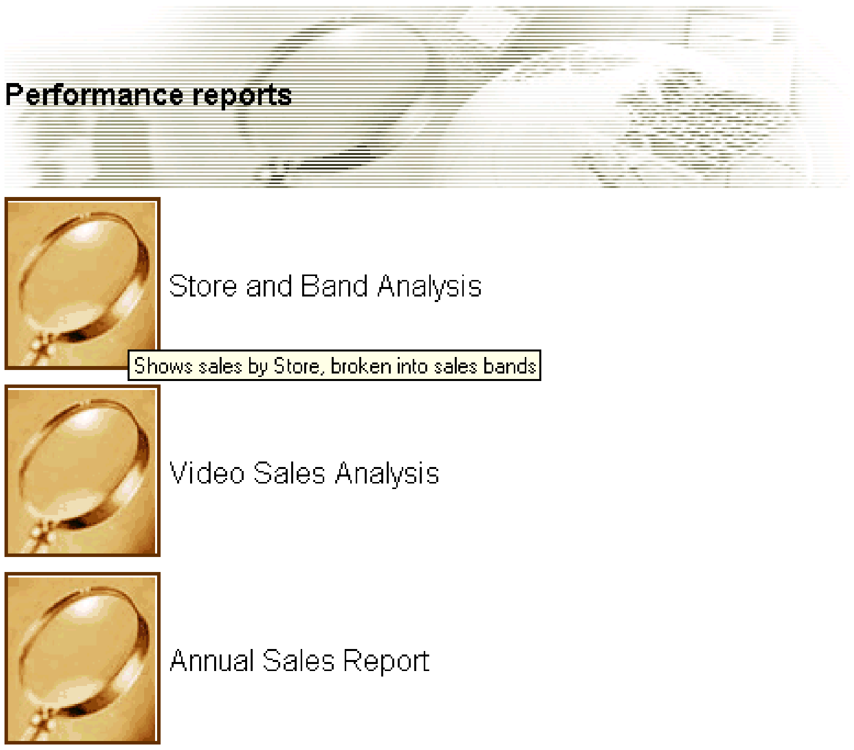
The XSL stylesheet is modified as follows:

```
<?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 7–3.

Figure 7–3 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 7–2.

Figure 7–4 shows a typical web-based rendition of this sample application.

Table 7–2 Explaining example2.xsl — After Modifying the XSL File

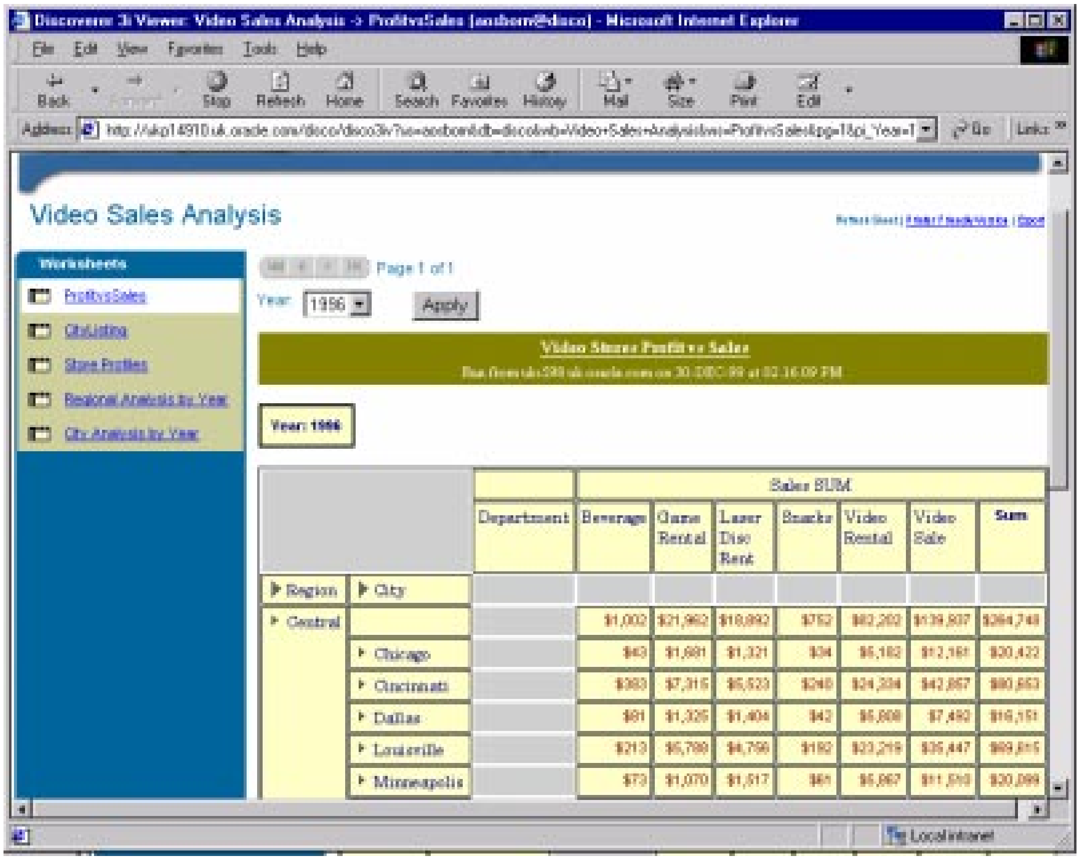
example2.xsl code	The code means ...

`<table border="0">`
` <tr>`
` <td width="500" height="100" background="disco_banner.gif">`
`   <font face="sans-serif">`
`     <b>Performance reports</b></font>`
`   </font>`
` </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>`
`      <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>`
	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 7–4 Discoverer 4i Viewer. Typical Web-Based Rendition as a Business Solution


Discoverer 3i Viewer: Video Sales Analysis -> ProfitvsSales [aosborn@disco] - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Stop Refresh Home Search Favorites History Mail Size Print Edit
Address
Go
Links
Worksheets
ProfitvsSales
CityListing
Store Profiles
Regional Analysis by Year
City Analysis by Year
Page 1 of 1
Year 1996
Apply
Local intranet

http://ukp14910.uk.oracle.com/disco/disco3iv?us=aosborn&db=disco&wb=Video+Sales+Analysis&ws=ProfitvsSales&pg=1&pi_Year=1

Video Sales Analysis

Video Stores Profit vs Sales

Run from uks598.uk.oracle.com on 30-DEC-99 at 02:16:09 PM

Year: 1996

			Sales SUM						
		Department	Beverage	Game Rental	Laser Disc Rent	Snacks	Video Rental	Video Sale	Sum
Region	City								
Central			$1,002	$21,962	$18,892	$752	$82,202	$139,807	$264,748
	Chicago		$43	$1,681	$1,321	$34	$6,182	$12,161	$20,422
	Cincinnati		$383	$7,316	$5,523	$240	$24,224	$42,857	$80,653
	Dallas		$81	$1,325	$1,404	$42	$5,808	$7,492	$16,151
	Louisville		$213	$5,788	$4,756	$192	$23,219	$35,447	$69,615
	Minneapolis		$73	$1,070	$1,517	$61	$5,867	$11,510	$20,099

Frequently Asked Questions (FAQs): Discoverer 4i 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 4i Viewer Communicates with Browsers

Question

What does Discoverer 4i Viewer use to communicate with the user’s browser?

Answer

HTTP and HTML.

Discoverer 4i Viewer and XML

Question

How is XML used by Discoverer 4i Viewer?

Answer

XML is generated by the middle-tier and represents the application state. Discoverer 4iViewer 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.

disco4iv.xml

Question

What does the disco4iv.xml file do?

Answer

You can use disco4iv.xml file to configure various options to make Discoverer 4i Viewer behave the way you want to. For example, you can specify the Discoverer Session that it should connect to.

Discoverer 4i and XSL

Question

How is XSL used by Discoverer 4i Viewer?

Answer

Discoverer 4i Viewer uses XSL (or more specifically XSLT) 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 XSLT Processors

Question

What XSL processors can be used by Discoverer 4i Viewer?

Answer

Discoverer 4i Viewer can be configured to use Oracle XSLTProcessor. This is the default and part of the XDK for Java.

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
- XML Spy

Customizing Stylesheets

Question

What is commonly changed in order to customize a stylesheet?

Answer

To customize a styelsheet, edit the following items:

- **disco4iv.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 4i 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
- **scripts.xsl** (not recommended) to change the JavaScript used to enhance the UI
- **render_table.xsl** (not recommended) to change how the table/crosstab is rendered

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 XSLT 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 disco4iv.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/1
- 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
- http://www.dpawson.co.uk/xsl/sect21.html

Discoverer Viewer XML’s DTD

Question

What is the structure of the Discoverer Viewer XML?

Answer

The XML documents generated by Discoverer Viewer conform to the following DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT account (error*, user?, database?, connect?, role*, eul*, option*,
version*)>
<!ATTLIST account
    name CDATA #IMPLIED
    mv_summaries_supported (true | false) "true"
    ref CDATA #IMPLIED
>

<!ELEMENT axes (axis)*>

<!ELEMENT axis (item*)>
<!ATTLIST axis
```

```
    position (m | x | y | z) #REQUIRED
>

<!ELEMENT background_color EMPTY>
<!ATTLIST background_color
    red CDATA #REQUIRED
    green CDATA #REQUIRED
    blue CDATA #REQUIRED
>

<!ELEMENT cell EMPTY>
<!ATTLIST cell
    result CDATA #REQUIRED
>

<!ELEMENT chart (page_item*, dim:image_map)>
<!ATTLIST chart
    name CDATA #REQUIRED
    height CDATA #REQUIRED
    width CDATA #REQUIRED
>

<!ELEMENT command (#PCDATA)>
<!ATTLIST command
    name CDATA #REQUIRED
    ref CDATA #IMPLIED
    implied (true | false) "false"
    valid (true | false) "true"
>

<!ELEMENT connect (#PCDATA)>

<!ELEMENT data (value, qdr?)>

<!ELEMENT database (#PCDATA)>

<!ELEMENT date (#PCDATA)>
<!ATTLIST date
    ref CDATA #IMPLIED
>

<!ELEMENT description (#PCDATA)>
<!ELEMENT error (#PCDATA | command)*>
<!ATTLIST error
code CDATA #REQUIRED
```

```
severity CDATA #IMPLIED
>

<!ELEMENT discoverer (session, request, account*, export*, locale?, version*)>
<!ATTLIST discoverer
    login_method (application | discoverer) "discoverer"
>

<!ELEMENT drill (#PCDATA)>
<!ATTLIST drill
    ref CDATA #IMPLIED
>

<!ELEMENT drill_path (#PCDATA)>
<!ATTLIST drill_path
    name CDATA #REQUIRED
    hierarchy_name CDATA #IMPLIED
    direction (collapse | up | down) #REQUIRED
    level CDATA #REQUIRED
    ref CDATA #REQUIRED
>

<!ELEMENT edge (item*, edge_row*)>
<!ATTLIST edge
    placement (page | side | top) #REQUIRED
>

<!ELEMENT edge_row (value*)>

<!ELEMENT eul (workbook*, version*)>
<!ATTLIST eul
    name CDATA #REQUIRED
    default (true | false) "false"
    ref CDATA #IMPLIED
>

<!ELEMENT export (#PCDATA)>
<!ATTLIST export
    name CDATA #REQUIRED
    ref CDATA #IMPLIED
    format CDATA #REQUIRED
>

<!ELEMENT font EMPTY>
<!ATTLIST font
```

```
    name CDATA #REQUIRED
    size CDATA #REQUIRED
    bold (true | false) "false"
    italic (true | false) "false"
    strikeout (true | false) "false"
    underline (true | false) "false"
>

<!ELEMENT foreground_color EMPTY>
<!ATTLIST foreground_color
    red CDATA #REQUIRED
    green CDATA #REQUIRED
    blue CDATA #REQUIRED
>

<!ELEMENT format (background_color, foreground_color, graphic_bar_color?, font)>
<!ATTLIST format
    id CDATA #REQUIRED
    description CDATA #IMPLIED
    display_name CDATA #IMPLIED
    horizontal_alignment (left | center | default | right) #REQUIRED
    vertical_alignment (bottom | center | top | lower_bound | upper_bound)
#REQUIRED
    graphic_bar_visible (true | false) "false"
    word_wrap (true | false) "false"
>

<!ELEMENT format_map (format*)>

<!ELEMENT graphic_bar_color EMPTY>
<!ATTLIST graphic_bar_color
    red CDATA #REQUIRED
    green CDATA #REQUIRED
    blue CDATA #REQUIRED
>

<!ELEMENT group (value*, group*, data*)>

<!ELEMENT item (drill_path*, sort*)>
<!ATTLIST item
    name CDATA #REQUIRED
    key CDATA #REQUIRED
    id CDATA #IMPLIED
    format_class CDATA #IMPLIED
    heading CDATA #IMPLIED
```

```
>

<!ELEMENT layout (row*)>

<!ELEMENT locale (#PCDATA)>
<!ATTLIST locale
    language CDATA #REQUIRED
    country CDATA #REQUIRED
>

<!ELEMENT measure_edge (item+)>
<!ATTLIST measure_edge
    placement CDATA #REQUIRED
    level CDATA #REQUIRED
>

<!ELEMENT option (#PCDATA)>
<!ATTLIST option
    name (aq | ftd | msa | nad | nv | qif | qll | qpw | qrl | qtl | rpp | usd)
#REQUIRED
    enable (true | false) "false"
>

<!ELEMENT page_item (drill_path*, sort*, value+)>
<!ATTLIST page_item
    name CDATA #REQUIRED
    ref CDATA #IMPLIED
    key CDATA #IMPLIED
    id CDATA #IMPLIED
    format_class CDATA #IMPLIED
    heading CDATA #IMPLIED
>

<!ELEMENT parameter (value, prompt)>
<!ATTLIST parameter
    name CDATA #REQUIRED
    ref CDATA #IMPLIED
    description CDATA #IMPLIED
    lov_exists (true | false) "false"
    multivalued (true | false) "false"
    wildcard_supported (true | false) "false"
    type CDATA #REQUIRED
>

<!ELEMENT prompt (#PCDATA)>
```

```
<!ELEMENT qdr (#PCDATA)>

<!ELEMENT query (parameter*, axes, sheet_data?, chart?, drill?)>
<!ATTLIST query
    version CDATA #REQUIRED
    status CDATA #REQUIRED
    step CDATA #REQUIRED
    elapsed CDATA #IMPLIED
    estimate CDATA #REQUIRED
>

<!ELEMENT request (error*, command*)>
<!ATTLIST request
    source CDATA #REQUIRED
    parameters CDATA #IMPLIED
>

<!ELEMENT role (#PCDATA | security_group)*>
<!ATTLIST role
    name CDATA #REQUIRED
    ref CDATA #IMPLIED
    current (true | false) "false"
>

<!ELEMENT row (cell*)>

<!ELEMENT security_group (#PCDATA)>

<!ELEMENT session EMPTY>
<!ATTLIST session
    id CDATA #REQUIRED
>

<!ELEMENT sheet_data (page_item*, format_map?, edge*, measure_edge?, group*,
error*)>
<!ATTLIST sheet_data
    name CDATA #REQUIRED
    row_range_begin CDATA #REQUIRED
    row_range_end CDATA #REQUIRED
    total_rows CDATA #REQUIRED
    mode (inline | outline) #REQUIRED
>

<!ELEMENT sheet_layout (axis+)>
```

```
<!ELEMENT sort EMPTY>
<!ATTLIST sort
    type (none | group | hidden | page) #REQUIRED
    direction (hi_lo | lo_hi) #REQUIRED
    line_width CDATA #IMPLIED
    spaces CDATA #IMPLIED
    level CDATA #IMPLIED
>

<!ELEMENT time (#PCDATA)>
<!ATTLIST time
    ref CDATA #IMPLIED
>

<!ELEMENT title (#PCDATA)>

<!ELEMENT user (#PCDATA)>

<!ELEMENT value (#PCDATA | drill_path)*>
<!ATTLIST value
    current (true | false) "false"
    default (true | false) "false"
    wildcard (true | false) "false"
    ref CDATA #IMPLIED
    format_class CDATA #IMPLIED
    item_class CDATA #IMPLIED
    type (item | spacing | total) #IMPLIED
    id CDATA #IMPLIED
    data CDATA #IMPLIED
    label CDATA #IMPLIED
>

<!ELEMENT version EMPTY>
<!ATTLIST version
    component CDATA #REQUIRED
    product CDATA #REQUIRED
    version CDATA #REQUIRED
>

<!ELEMENT workbook (description?, worksheet*, date?, time?)>
<!ATTLIST workbook
    name CDATA #REQUIRED
    ref CDATA #IMPLIED
>
```

```
<!ELEMENT worksheet (description?, sheet_layout?, title?, layout?, query*)>
<!ATTLIST worksheet
    name CDATA #REQUIRED
    ref CDATA #REQUIRED
>

<!ELEMENT dim:image_map (GraphMap)>
<!ATTLIST dim:image_map
    xmlns:dim CDATA #REQUIRED
>

<!--
The GraphMap and related entities are provided by the BI Beans team.
-->
<!ELEMENT GraphMap (DataLine | DataMarker | TwoDMarker | StockMarker |
AreaMarker | ThreeDMarker | LegendMarker |
    LegendText | MarkerText | PieLabel | Slice | SliceLabel | O1TickLabel |
X1TickLabel | O1Title | X1Title |
    Y1TickLabel | Y1Title | Y2TickLabel | Y2Title | ZTickLabel | ZTitle | Title
| Subtitle | Footnote)*>

<!ELEMENT DataLine (Group, Series, Tooltip?, Geometry)>

<!ELEMENT DataMarker (Group, Series, Tooltip?, Geometry)>
<!ELEMENT TwoDMarker (Group, Series, Tooltip?, Geometry)>
<!ELEMENT StockMarker (Group, Series, Tooltip?, Geometry)>
<!ELEMENT AreaMarker (Group, Series, Tooltip?, Geometry)>
<!ELEMENT ThreeDMarker (Group, Series, Tooltip?,Geometry)>
<!ELEMENT LegendMarker (Series, Geometry)>
<!ELEMENT LegendText (Series, Geometry)>
<!ELEMENT MarkerText (Group, Series, Geometry)>
<!ELEMENT PieLabel (Group, Geometry)>
<!ELEMENT Slice (Group, Series, Tooltip?, Geometry)>
<!ELEMENT SliceLabel (Group, Series, Geometry)>
<!ELEMENT O1TickLabel (Group, Geometry)>
<!ELEMENT X1TickLabel (Geometry)>
<!ELEMENT O1Title (Geometry)>
<!ELEMENT X1Title (Geometry)>
<!ELEMENT Y1TickLabel (Geometry)>
<!ELEMENT Y1Title (Geometry)>
<!ELEMENT Y2TickLabel (Geometry)>
<!ELEMENT Y2Title (Geometry)>
<!ELEMENT ZTickLabel (Series, Geometry)>
<!ELEMENT ZTitle (Series, Geometry)>
```

```
<!ELEMENT Title (Geometry)>
<!ELEMENT Subtitle (Geometry)>
<!ELEMENT Footnote (Geometry)>
<!ELEMENT Group (#PCDATA)>
<!ELEMENT Series (#PCDATA)>
<!ELEMENT Tooltip (Line)*>
<!ELEMENT Line (#PCDATA)>
<!ELEMENT Geometry (Vertex)*>
<!ELEMENT Vertex EMPTY>
<!ATTLIST Vertex x CDATA #REQUIRED>
<!ATTLIST Vertex y CDATA #REQUIRED>
```

8

Online B2B XML Application: Step by Step

This chapter contains the following topics:

Introduction to the Online B2B XML Application

This chapter describes all the steps and scripts you need to build an online B2B XML application.

A modified version of this application is available on Oracle Technology Network (OTN) site: http://otn.oracle.com/tech/xml, under "WebStore B2B Demo". This modifed version adds support for multiple Suppliers and Retailers, and can be expanded for a larger B2B online data exchange than is described in this chaper. You can also download the scripts from this OTN site.

Requirements for Running the Online B2B XML Application

The following lists requirements to build and run the online 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, for exmple.
  - 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 Online B2B XML Application: Overview

This XML application and demo illustrate 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.
- The Retailer and Supplier can then view 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* or higher Database(s)**. This solution assumes both the Retailer and Supplier are storing their data in Oracle8*i* or higher 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 8–2.

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 8–2, "Online 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 XSLT Processor
- Oracle8*i* Release 3 (8.1.7) or higher or Oracle9*i*

Tools Used

JDeveloper

Note: No pre-authored (static or composed) XML documents are used in this B2B XML application. All XML documents in this application are dynamically generated (decomposed) from the database.

Why Transform Data to XML?

Retailers and Suppliers use many 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 8–1 illustrates the overall flow of the Retailer-Supplier transaction. The Retailer enters the order.

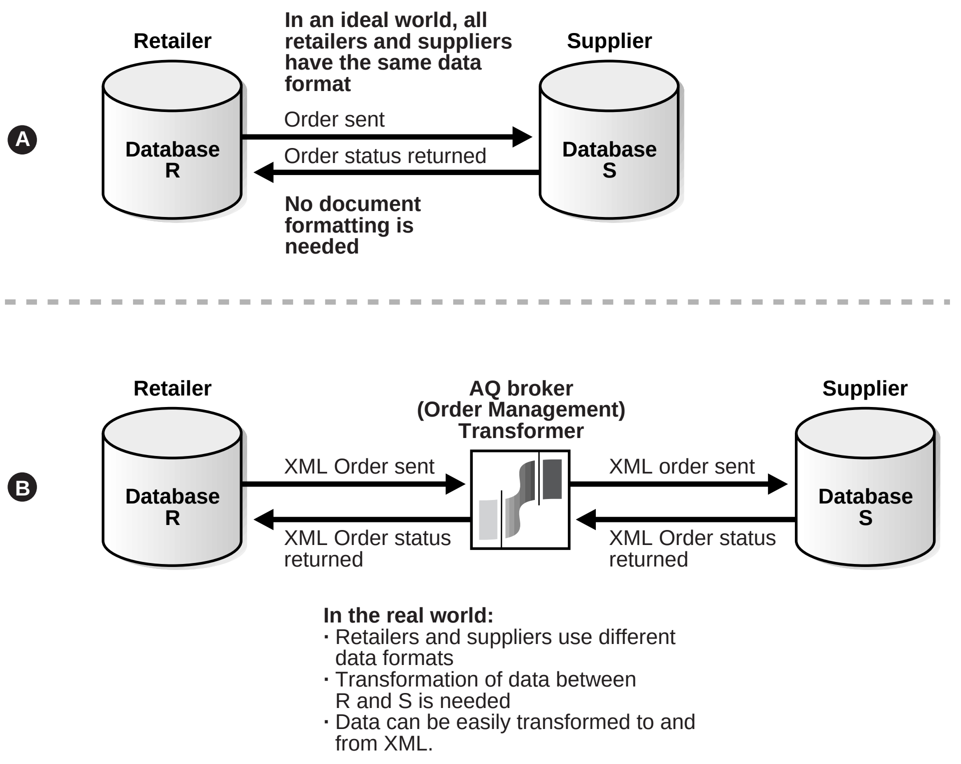
- In an ideal world, if the order document format of every Retailer and every Supplier were the same, the process would be simply as shown in A.
- In the real world, order document formats of each Retailer and Supplier typically differ. Making these transactions seamless is possible by converting the 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

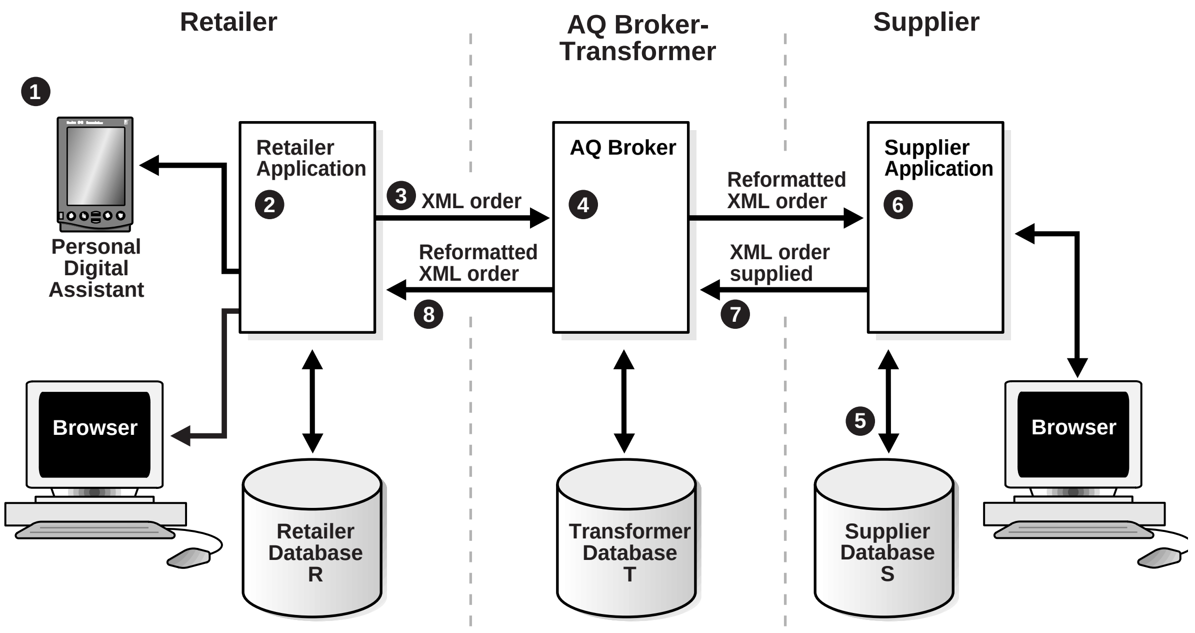
Figure 8–1 Why Transform Data to XML?: Retailer's Order Data Can be recognized by Any Supplier - Supplier's Order Status and Acknowledgement Can be Recognized by any Retailer


A
Retailer
Database R
In an ideal world, all retailers and suppliers have the same data format
Order sent
Order status returned
No document formatting is needed
Supplier
Database S
B
Retailer
Database R
XML Order sent
XML Order status returned
AQ broker (Order Management) Transformer
XML order sent
XML Order status returned
Supplier
Database S
In the real world:
· Retailers and suppliers use different data formats
· Transformation of data between R and S is needed
· Data can be easily transformed to and from XML.

Online B2B XML Application: Main Components

Figure 8–2 shows the main components used in this online 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 8–5.

Figure 8–2 Online B2B XML Application: Main Components



Overview of Tasks to Run the Online B2B XML Application

The schemas used in the B2B XML application you are about to build, are illustrated in Figure 8–3.

To run the B2B XML application carry out the following tasks as described:

- Task 1. Set Up Your Environment to Run the Online B2B XML Application
- Task 2. Run the B2B Application
- Task 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 8-34. You will also see typical screenshots of what the Retailer and Supplier see.

Figure 8–3   B2B XML Retailer (Customers) and Supplier Schema

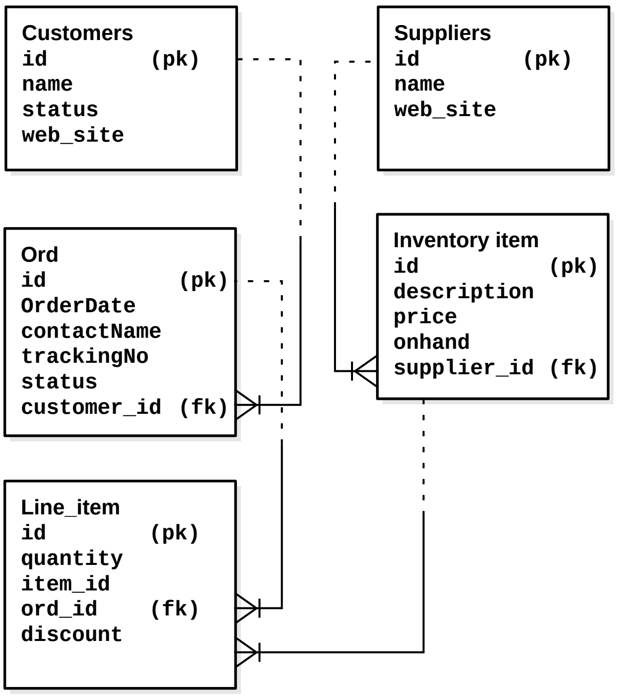
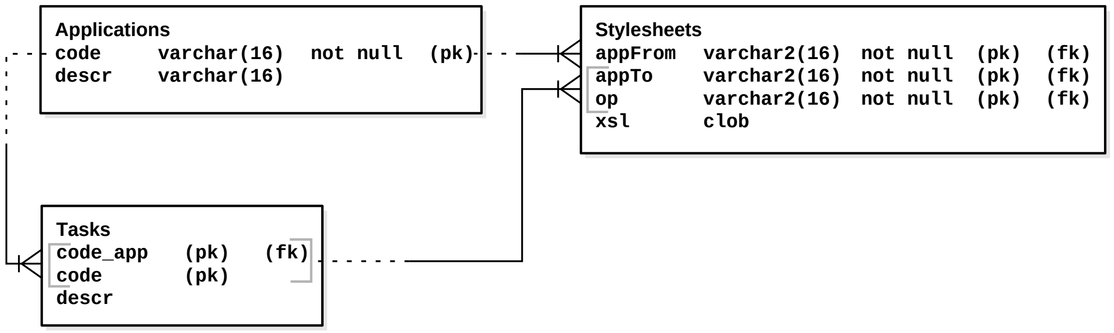


Figure 8–4   B2B XML AQ Broker Schema: Stylesheets


Applications
code varchar(16) not null (pk)
descr varchar(16)
Stylesheets
appFrom varchar2(16) not null (pk) (fk)
appTo varchar2(16) not null (pk) (fk)
op varchar2(16) not null (pk) (fk)
xsl clob
Tasks
code_app (pk) (fk)
code (pk)
descr

Task 1. Set Up Your Environment to Run the Online 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 "Online 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.

Task 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".

Task 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.

Online B2B XML 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 Code Calling Sequence

The following list provides the SQL example code 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 8–3 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 supplier 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,
 'BackOrdered', 0);
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 8–4. 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

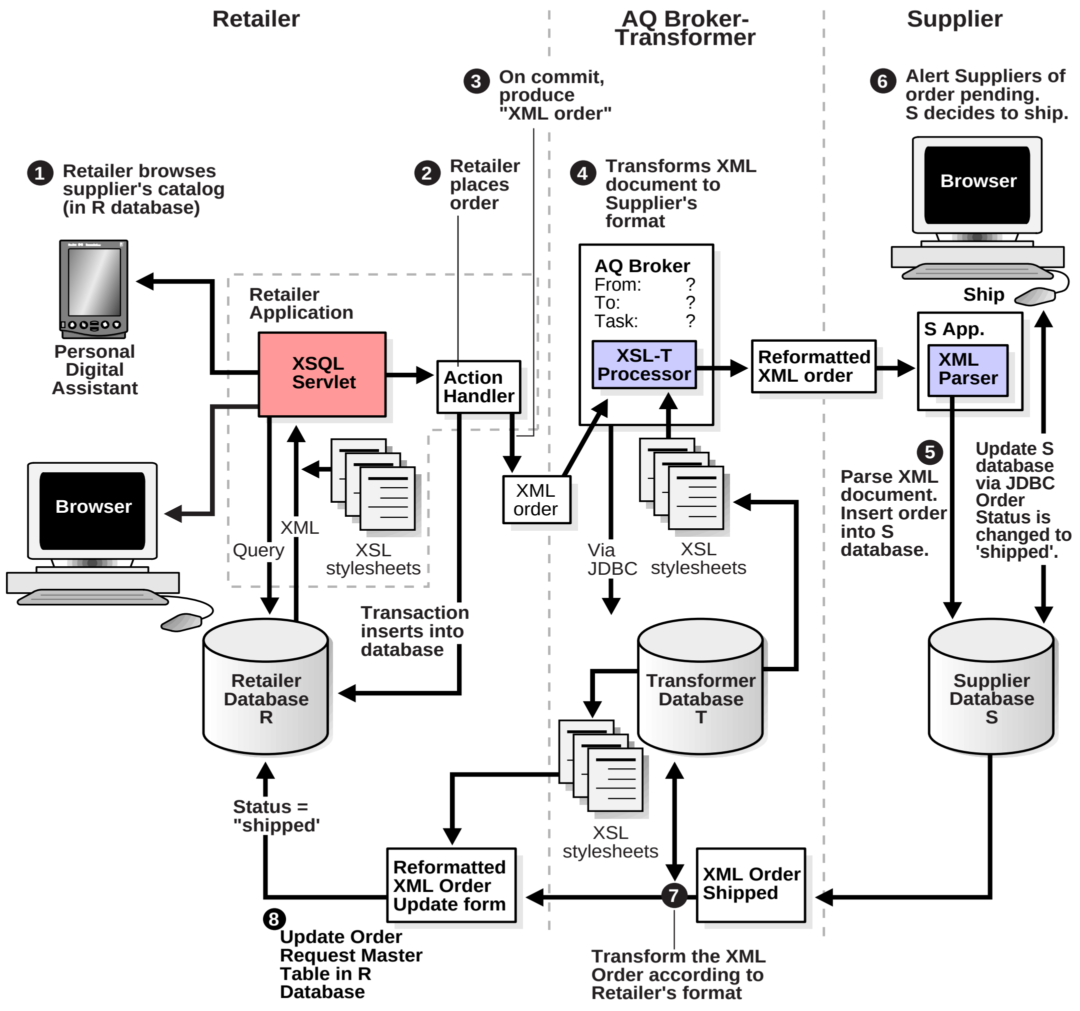
This SQL script deletes orders from the Retailer-Supplier database Customers table according to the customer’s ID.

```
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'))
/
```

Online B2B XML Application: Data Exchange Flow

Figure 8–5 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 8–5 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
AQ Broker-Transformer
Supplier
3 On commit, produce "XML order"
6 Alert Suppliers of order pending. S decides to ship.
1 Retailer browses supplier's catalog (in R database)
2 Retailer places order
4 Transforms XML document to Supplier's format
Browser
Ship
Personal Digital Assistant
Retailer Application
XSQL Servlet
Action Handler
AQ Broker
From: ?
To: ?
Task: ?
XSL-T Processor
Reformatted XML order
S App.
XML Parser
Browser
XML
Query
XSL stylesheets
XML order
Via JDBC
XSL stylesheets
5
Parse XML document. Insert order into S database.
Update S database via JDBC Order Status is changed to 'shipped'.
Transaction inserts into database
Retailer Database R
Transformer Database T
Supplier Database S
XSL stylesheets
Status = "shipped'
Reformatted XML Order Update form
7
XML Order Shipped
8 Update Order Request Master Table in R Database
Transform the XML Order according to Retailer's format

Retailer-Supplier Transactions

Figure 8–5 shows the business flow of the Retailer - Supplier transactions. These transactions are summarized here.

- Step 1. Retailer Browses the Supplier's OnLine "Hi-Tech Mall" Catalog
- Step 2. Retailer Places Order
- Step 3. Retailer Confirms and Commits to Sending the Order
- Step 4. AQ Broker-Transformer Transforms the XML Document According to the Supplier's Format
- Step 5. Supplier Application Parses Incoming Reformatted XML Order Document. Inserts Order into the Supplier Database
- Step 6. Supplier Application Alerts Supplier of Pending Order
- Step 7. AQ Broker-Transformer Transforms the XML Order According to Retailer's Format
- Step 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 8-34.

Step 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 XSQL.
2. Retailer browses the Supplier's on-line catalog. Retailer selects a product and quantity.

Step 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.

Step 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.

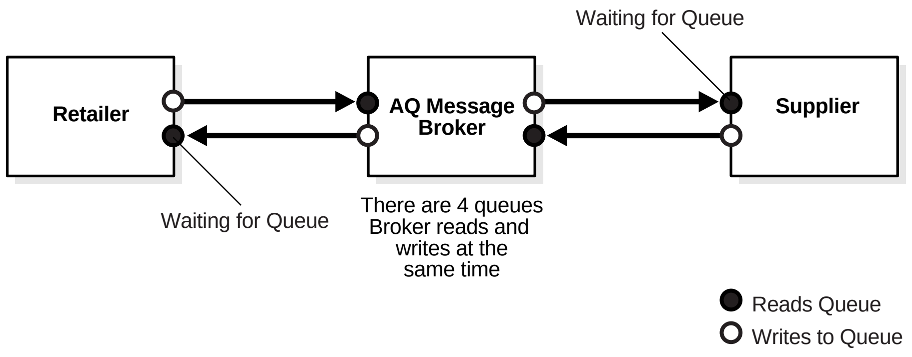
The Action Handler also sends the XML order on to the AQ Broker-Transformer.

Step 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 8–6.

Figure 8–6 Online 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. XSLT 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 XSLT 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 inFigure 8–4.

Step 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.

Step 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".

Step 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.

Step 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 8–5 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 Online B2B XML Application".

The following steps explain the process and how to run this application.

- Step 1. Retailer Browses the Supplier's OnLine "Hi-Tech Mall" Catalog
- Step 2. Retailer Places Order
- Step 3. "Validate" Commits the Transaction. Retailer Application Produces the XML Order
- Step 4. AQ Broker-Transformer Transforms XML Document According to Supplier's Format
- Step 5. Supplier Application Parses the XML Document and Inserts the Order into the Supplier Database
- Step 6a. Supplier Application Alerts Supplier of Pending Order
- Step 7. AQ Broker-Transformer Transforms XML Order into Retailer's Format

- Step 8. Retailer Application Updates the Ord Table and Displays the New Order Status to Retailer

Step 1. Retailer Browses the Supplier's OnLine "Hi-Tech Mall" Catalog

See Figure 8–5 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.

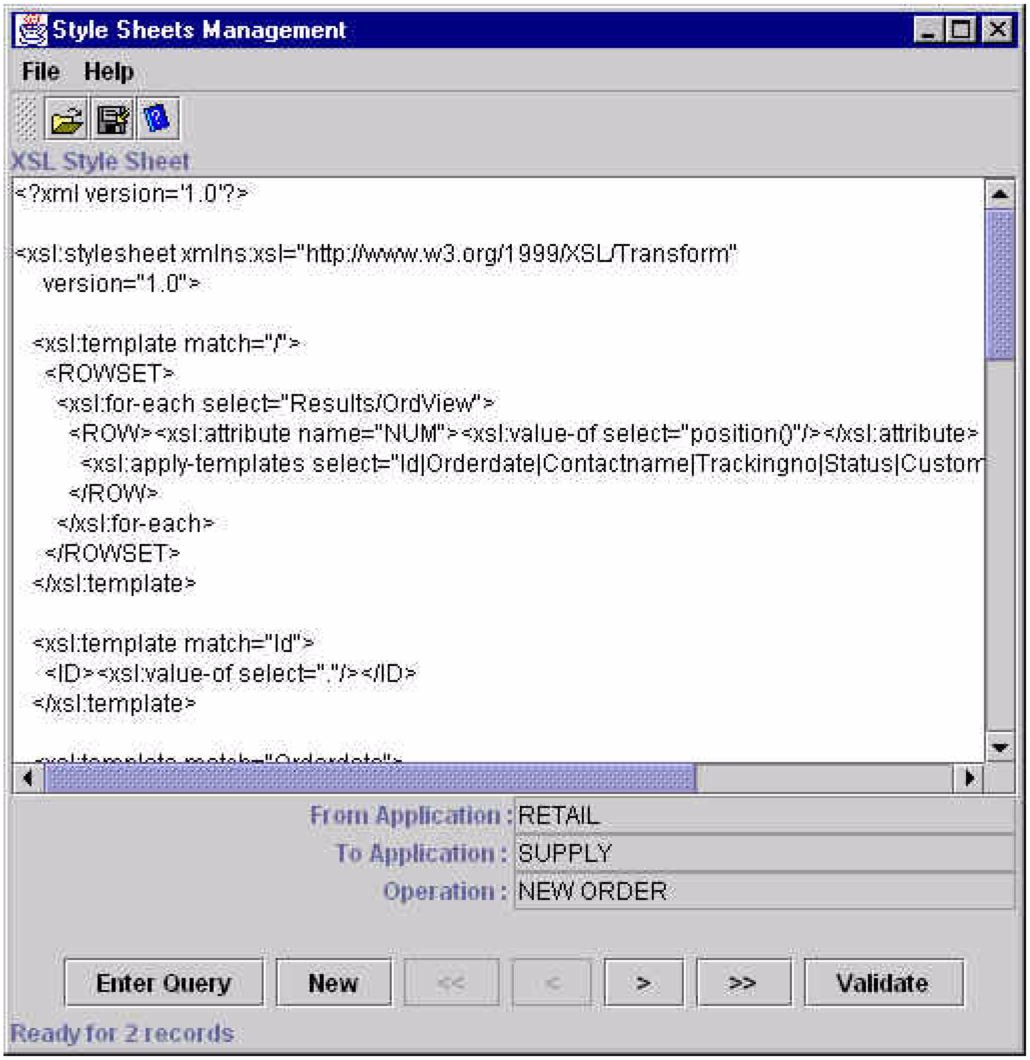
1. Check the StyleSheet utility to ensure it works by invoking `SS.bat`.

   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 8–7.

Figure 8–7 Checking the StyleSheet Utility



```
<?xml version='1.0'?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0">

  <xsl:template match="/">
   <ROWSET>
    <xsl:for-each select="Results/OrdView">
     <ROW><xsl:attribute name="NUM"><xsl:value-of select="position()"/></xsl:attribute>
       <xsl:apply-templates select="Id|Orderdate|Contactname|Trackingno|Status|Custom
     </ROW>
    </xsl:for-each>
   </ROWSET>
  </xsl:template>

  <xsl:template match="Id">
   <ID><xsl:value-of select="."/></ID>
  </xsl:template>
```

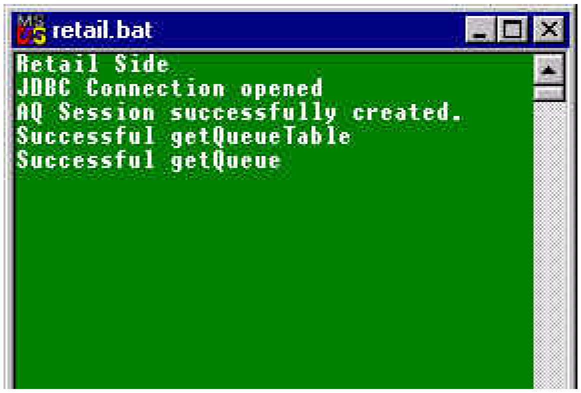
From Application : RETAIL

To Application : SUPPLY

Operation : NEW ORDER

2. Start the Retailer application by running retailer.bat. See Figure 8–8.

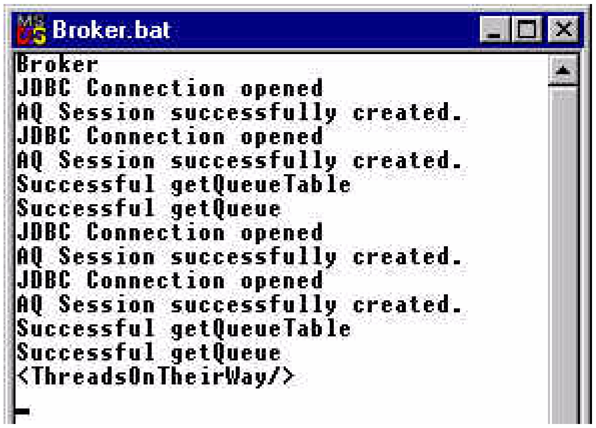
Figure 8–8 Starting the Retailer Application



```
Retail Side
JDBC Connection opened
AQ Session successfully created.
Successful getQueueTable
Successful getQueue
```

3. Start the AQ Broker-Transformer application by running broker.bat. See Figure 8–9.

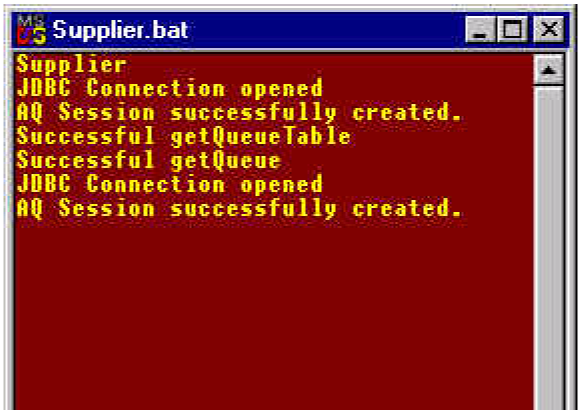
Figure 8–9 Starting the AQ Broker-Transformer Application



```
Broker
JDBC Connection opened
AQ Session successfully created.
JDBC Connection opened
AQ Session successfully created.
Successful getQueueTable
Successful getQueue
JDBC Connection opened
AQ Session successfully created.
JDBC Connection opened
AQ Session successfully created.
Successful getQueueTable
Successful getQueue
<ThreadsOnTheirWay/>
```

4. Start the Supplier application by running supplier.bat. See Figure 8–10.

Figure 8–10   Starting the Supplier Application: "Supplier Watcher"



```
Supplier
JDBC Connection opened
AQ Session successfully created.
Successful getQueueTable
Successful getQueue
JDBC Connection opened
AQ Session successfully created.
```

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 8–11.

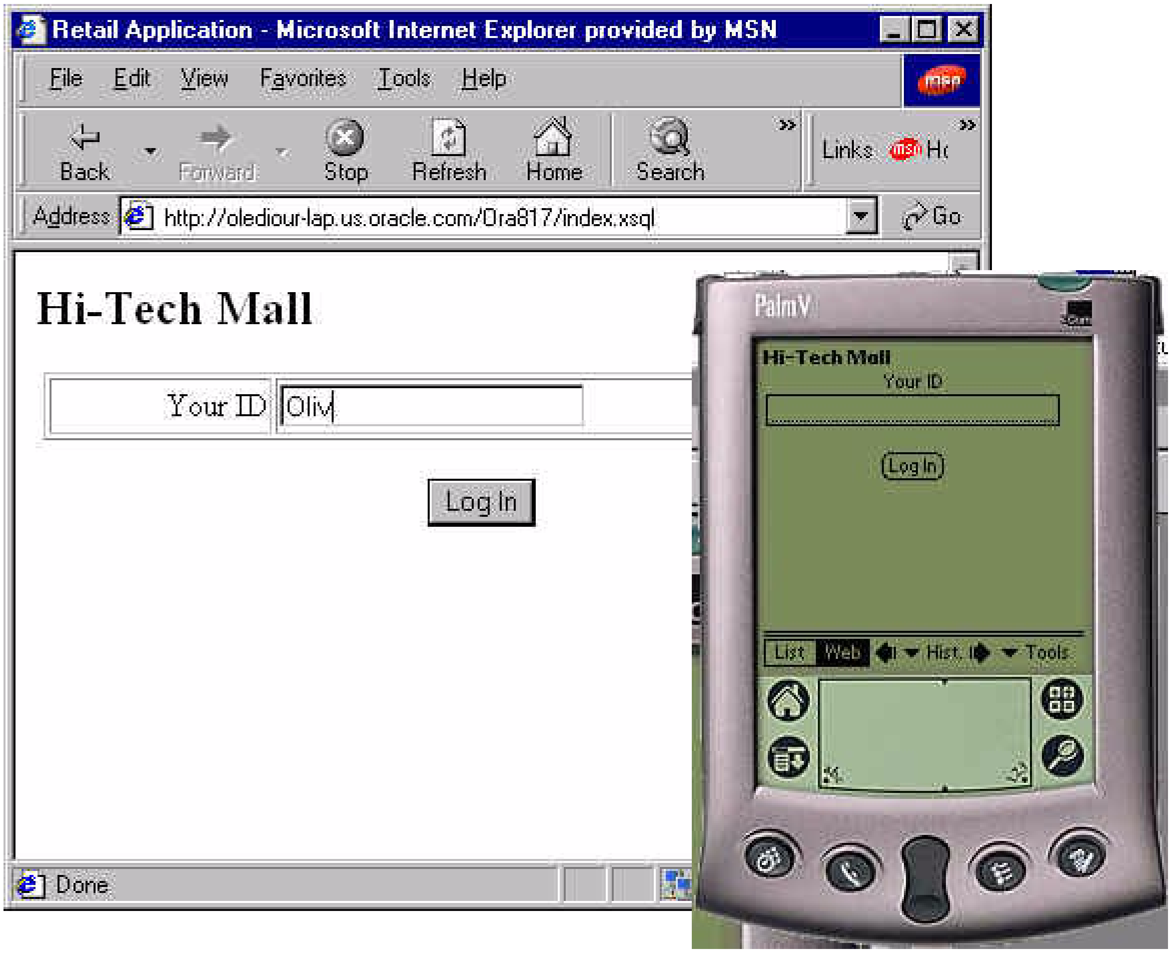
XSQL Script Example 2: Checking the ID of Users Logging In: getlogged.xsql

```
<?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 8–11   [Retailer]: Logging in from a Browser or PDA


Retail Application - Microsoft Internet Explorer provided by MSN
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Links
Address http://olediour-lap.us.oracle.com/Ora817/index.xsql
Go
Hi-Tech Mall
Your ID Oliv
Log In
Done
PalmV
Hi-Tech Mall
Your ID
Log In
List Web Hist. Tools

7. [Retailer]: Click on 'Please Enter the Mall'.

XSQL Script Example 1: Displays First Hi-Tech Mall Screen — index.xsql

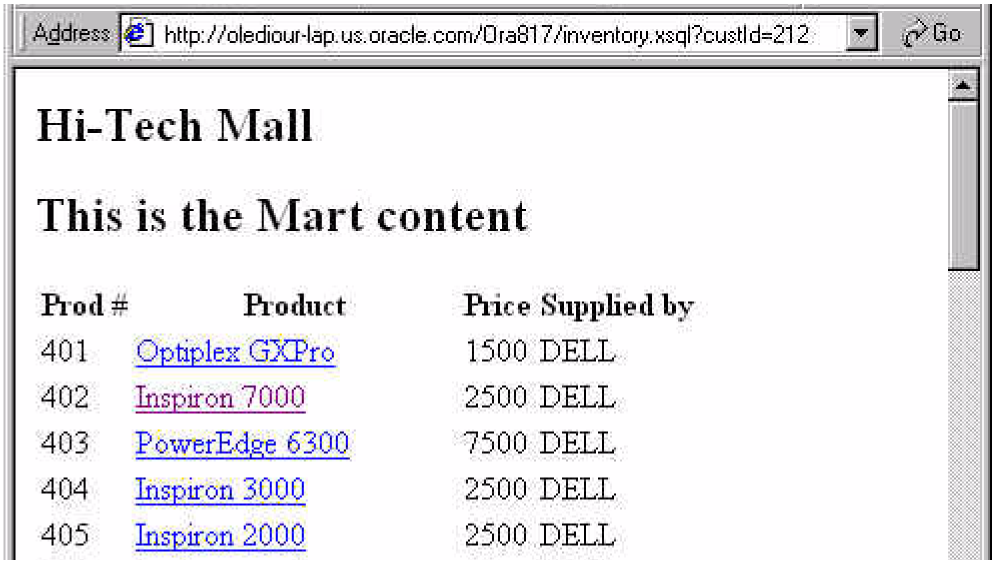
```
<?xml version="1.0"?>

<!--
 | 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 8–12.

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 8–12   [Retailer] Enter the Hi-Tech Mall (Mart) Catalog



Address http://olediour-lap.us.oracle.com/Ora817/inventory.xsql?custId=212

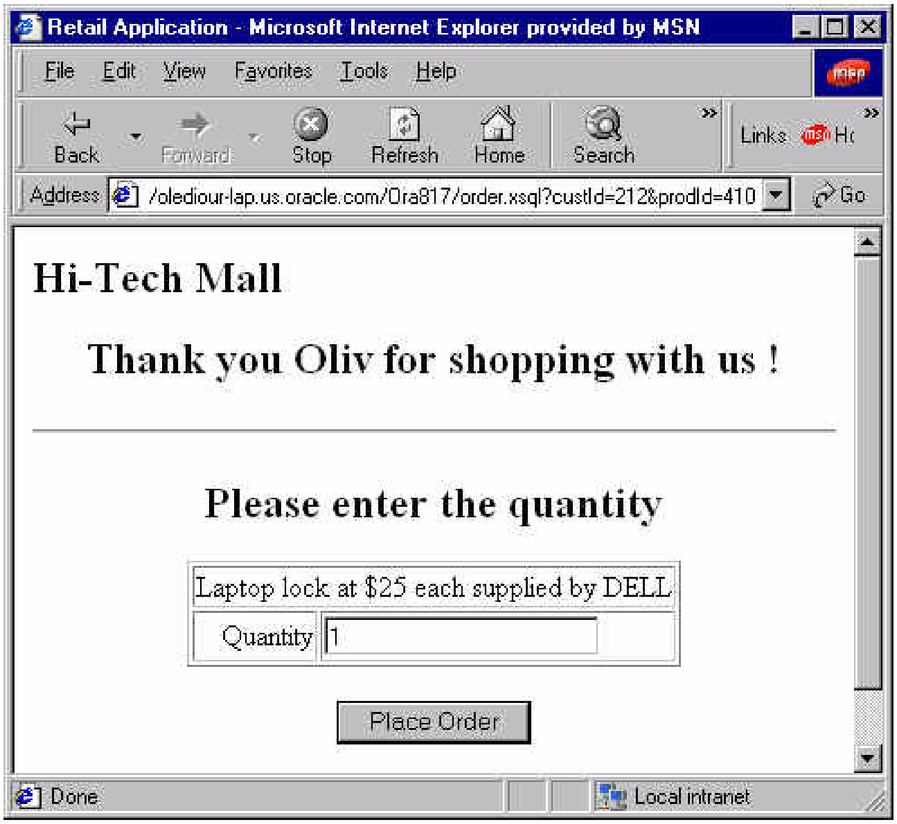
Hi-Tech Mall

This is the Mart content

Prod #	Product	Price	Supplied by
401	Optiplex GXPro	1500	DELL
402	Inspiron 7000	2500	DELL
403	PowerEdge 6300	7500	DELL
404	Inspiron 3000	2500	DELL
405	Inspiron 2000	2500	DELL

9. [Retailer]: Enter the quantity you need and click the "Place Order" button. See Figure 8–13.

Figure 8–13 [Retailer]: Enter the Quantity and Click on "Place Order"


Retail Application - Microsoft Internet Explorer provided by MSN
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Links
Address /olediour-lap.us.oracle.com/Ora817/order.xsql?custId=212&prodId=410
Go
Place Order
Done
Local intranet

Hi-Tech Mall

Thank you Oliv for shopping with us !

Please enter the quantity

Laptop lock at $25 each supplied by DELL	
Quantity	1

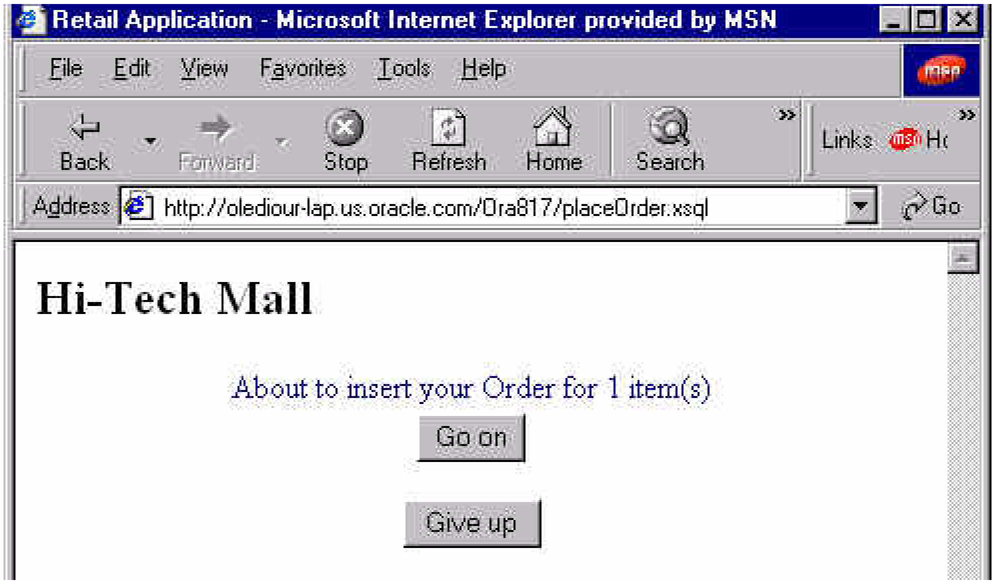
10. [Retailer] Click "Go On", or "Give Up". See Figure 8–14.

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 8–14 [Retailer}: Click "Go On"


Retail Application - Microsoft Internet Explorer provided by MSN
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Links
Address http://olediour-lap.us.oracle.com/Ora817/placeOrder.xsql
Go

Hi-Tech Mall

About to insert your Order for 1 item(s)

Step 2. Retailer Places Order

The Retailer selects "Go On", then the application checks the order, perhaps the retailer’s credit history, and then validates the order by selecting "Validate". See Figure 8–15 and Figure 8–16.

Figure 8–15 [Retailer]: Click "Validate"

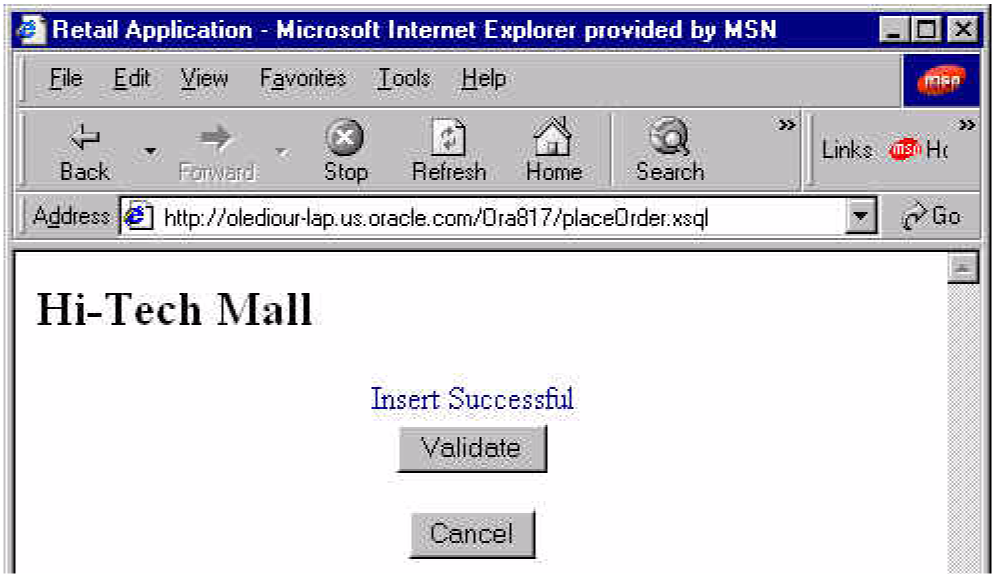
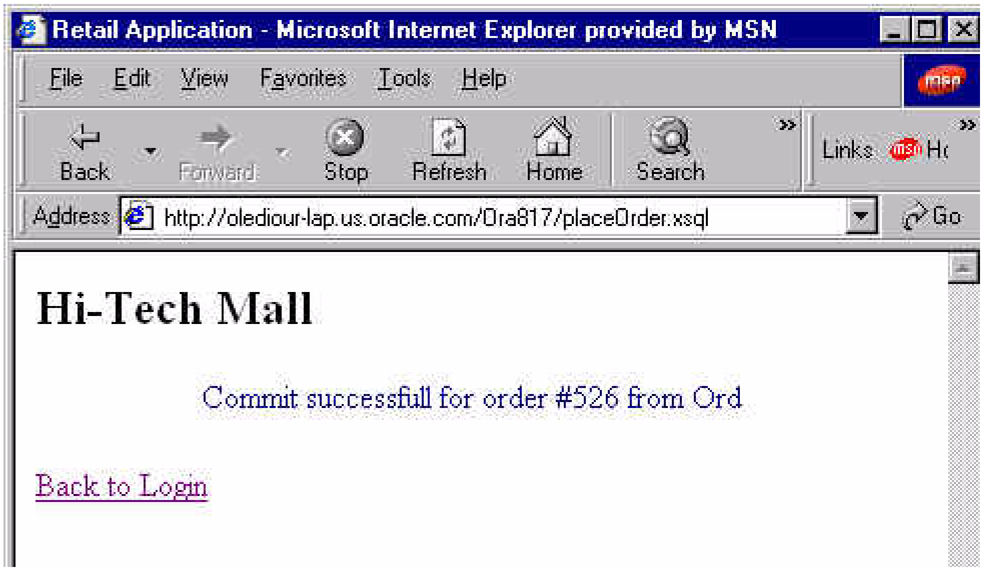

Retail Application - Microsoft Internet Explorer provided by MSN
File
Edit
View
Favorites
Tools
Help
Back
Forward
Stop
Refresh
Home
Search
Links
Address
http://olediour-lap.us.oracle.com/Ora817/placeOrder.xsql
Go
Hi-Tech Mall
Insert Successful
Validate
Cancel

Figure 8–16 [Retailer]: Commit Successful. Table Ord has Been Updated


Retail Application - Microsoft Internet Explorer provided by MSN
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Links
Address http://olediour-lap.us.oracle.com/Ora817/placeOrder.xsql
Go

Hi-Tech Mall

Commit successfull for order #526 from Ord

Back to Login

Step 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);
      }
    }
  }
}
```

Step 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

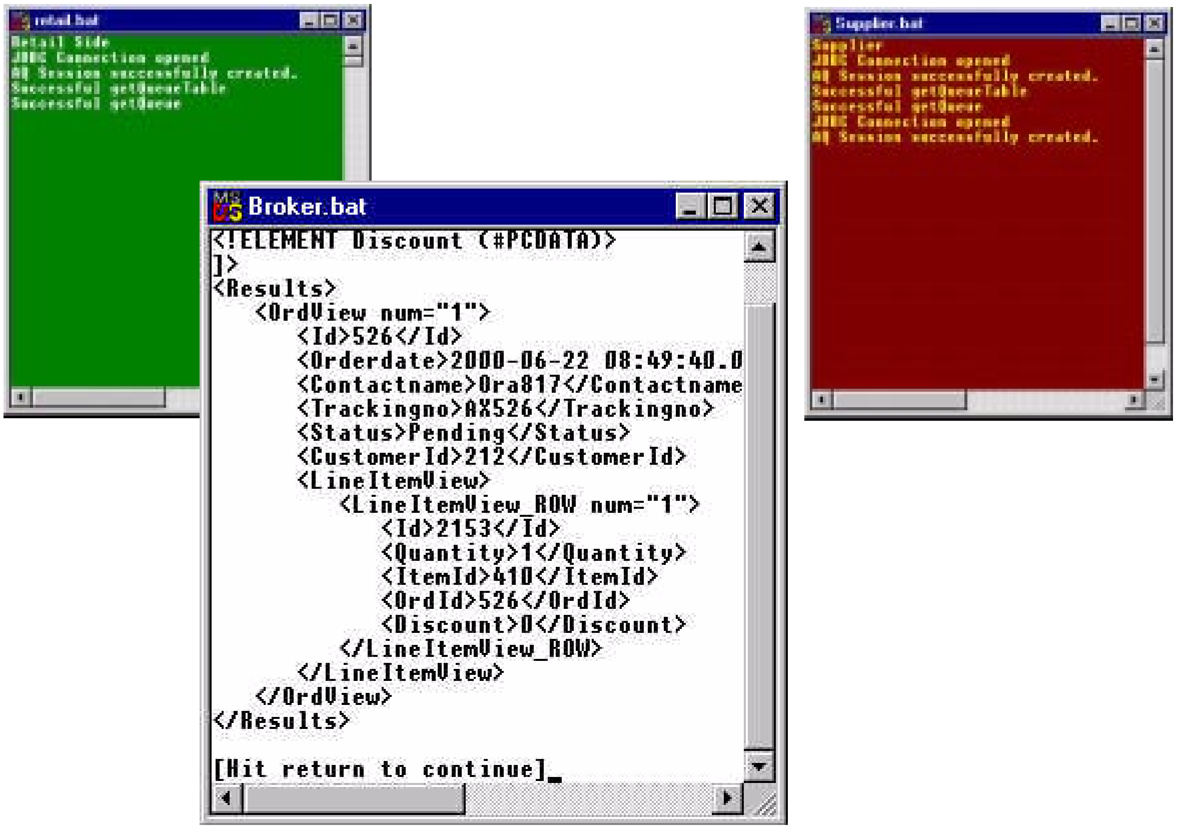
  These elements are used to select the correct stylesheet from Stylesheet table. XSLT Processor processes the transformation. See Figure 8–17.

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 8–17 [AQ Broker]: Viewing the retailer.bat, broker.bat, and supplier.bat Consoles (1 of 3)



```
Retail Side
JDBC Connection opened
AQ Session successfully created.
Successful getQueueTable
Successful getQueue
```

```
<!ELEMENT Discount (#PCDATA)>
]>
<Results>
   <OrdView num="1">
      <Id>526</Id>
      <Orderdate>2000-06-22 08:49:40.0
      <Contactname>Ora817</Contactname
      <Trackingno>AX526</Trackingno>
      <Status>Pending</Status>
      <CustomerId>212</CustomerId>
      <LineItemView>
         <LineItemView_ROW num="1">
            <Id>2153</Id>
            <Quantity>1</Quantity>
            <ItemId>410</ItemId>
            <OrdId>526</OrdId>
            <Discount>0</Discount>
         </LineItemView_ROW>
      </LineItemView>
   </OrdView>
</Results>

[Hit return to continue]_
```

```
Supplier
JDBC Connection opened
AQ Session successfully created.
Successful getQueueTable
Successful getQueue
JDBC Connection opened
AQ Session successfully created.
```

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.

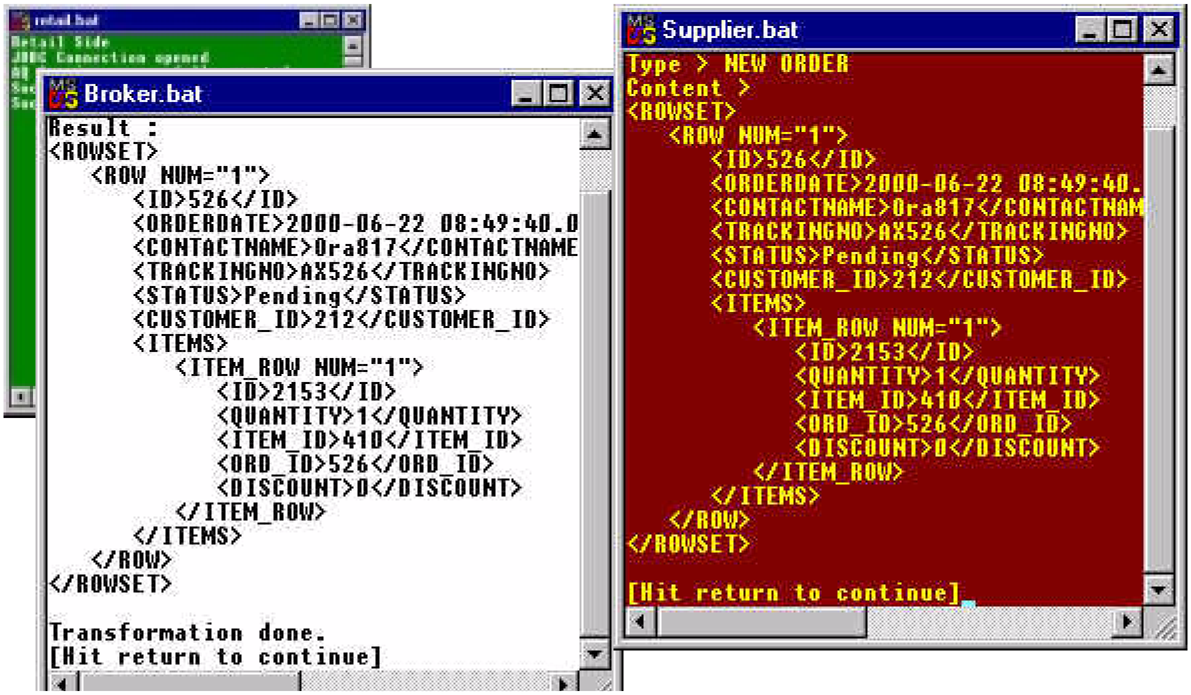
Note: Here XML + XSL = XML

---

5. Again hit [Return] in the AQ Broker Console. See Figure 8–18. The broker.bat screen changes as it has been transforming. The result is obtained after the XSLT 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 8–18 [AQ Broker]: Viewing the retailer.bat, broker.bat, and supplier.bat Consoles (2 of 3)



```
Retail Side
JDBC Connection opened
```

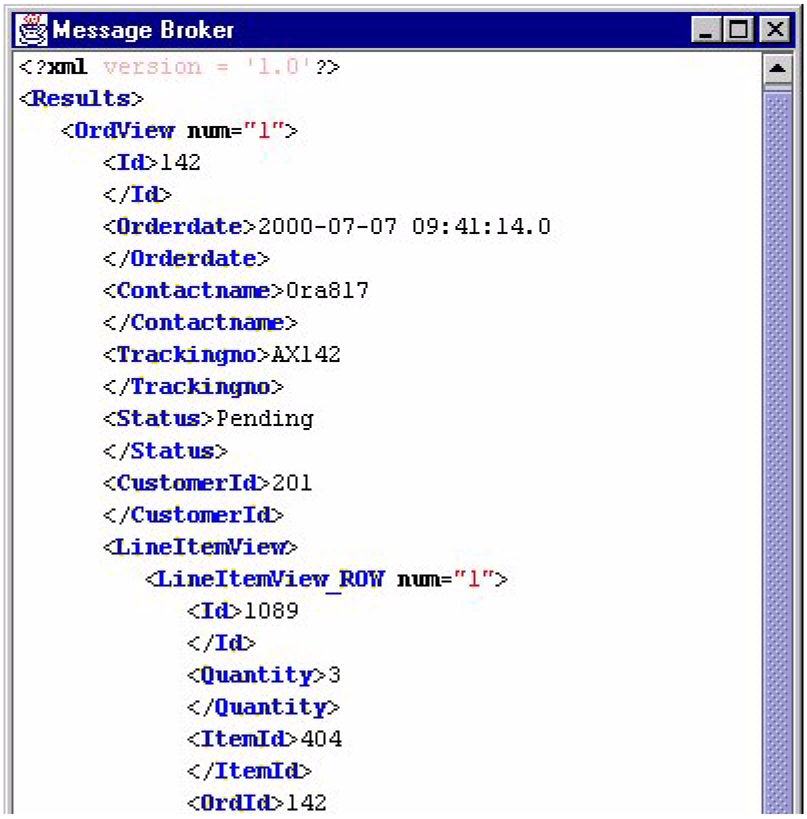
```
Result :
<ROWSET>
   <ROW NUM="1">
      <ID>526</ID>
      <ORDERDATE>2000-06-22 08:49:40.0
      <CONTACTNAME>Ora817</CONTACTNAME
      <TRACKINGNO>AX526</TRACKINGNO>
      <STATUS>Pending</STATUS>
      <CUSTOMER_ID>212</CUSTOMER_ID>
      <ITEMS>
         <ITEM_ROW NUM="1">
            <ID>2153</ID>
            <QUANTITY>1</QUANTITY>
            <ITEM_ID>410</ITEM_ID>
            <ORD_ID>526</ORD_ID>
            <DISCOUNT>0</DISCOUNT>
         </ITEM_ROW>
      </ITEMS>
   </ROW>
</ROWSET>

Transformation done.
[Hit return to continue]
```

```
Type > NEW ORDER
Content >
<ROWSET>
   <ROW NUM="1">
      <ID>526</ID>
      <ORDERDATE>2000-06-22 08:49:40.
      <CONTACTNAME>Ora817</CONTACTNAM
      <TRACKINGNO>AX526</TRACKINGNO>
      <STATUS>Pending</STATUS>
      <CUSTOMER_ID>212</CUSTOMER_ID>
      <ITEMS>
         <ITEM_ROW NUM="1">
            <ID>2153</ID>
            <QUANTITY>1</QUANTITY>
            <ITEM_ID>410</ITEM_ID>
            <ORD_ID>526</ORD_ID>
            <DISCOUNT>0</DISCOUNT>
         </ITEM_ROW>
      </ITEMS>
   </ROW>
</ROWSET>

[Hit return to continue]
```

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 8–19 Sample XML Document Output From AQ Broker-Transformer

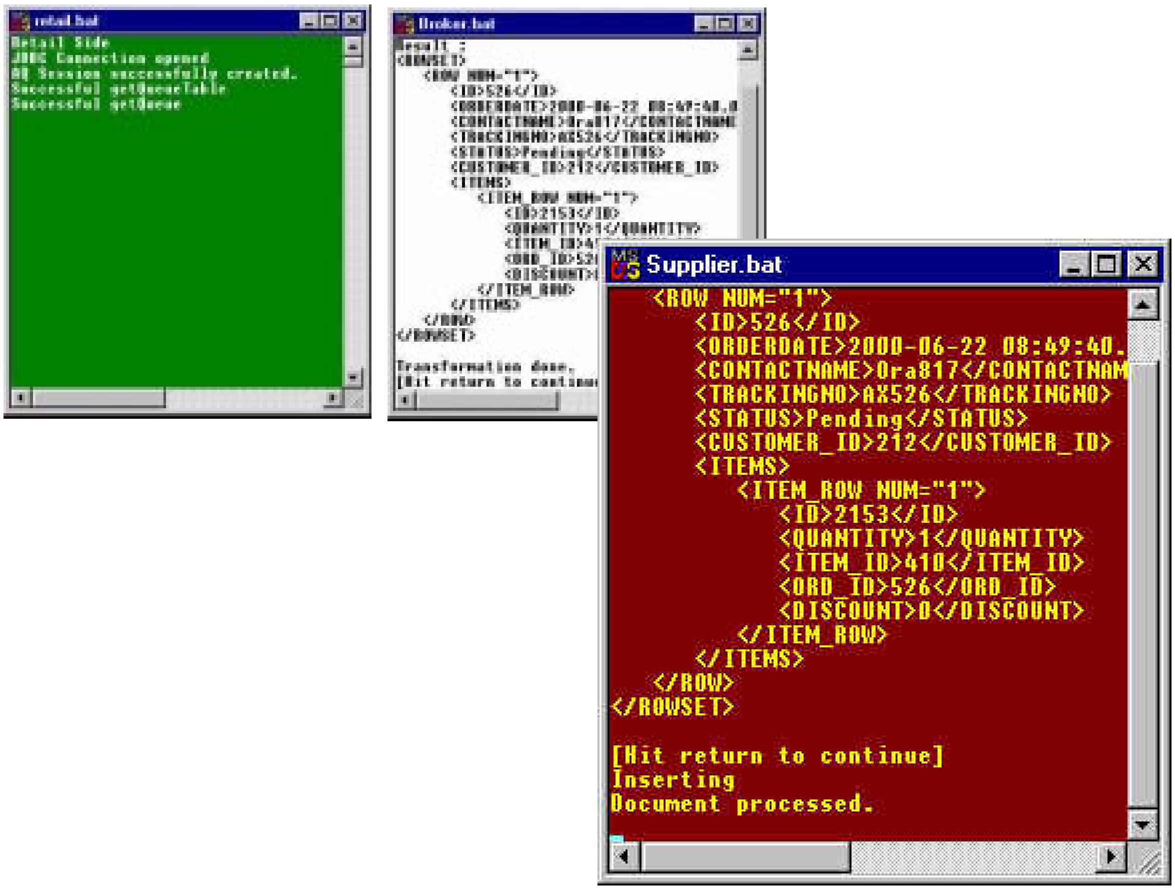


```
<?xml version = '1.0'?>
<Results>
   <OrdView num="1">
      <Id>142
      </Id>
      <Orderdate>2000-07-07 09:41:14.0
      </Orderdate>
      <Contactname>Ora817
      </Contactname>
      <Trackingno>AX142
      </Trackingno>
      <Status>Pending
      </Status>
      <CustomerId>201
      </CustomerId>
      <LineItemView>
         <LineItemView_ROW num="1">
            <Id>1089
            </Id>
            <Quantity>3
            </Quantity>
            <ItemId>404
            </ItemId>
            <OrdId>142
```

Step 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 8–20.

Figure 8–20 [AQ Broker]: Viewing the retailer.bat, broker.bat, and supplier.bat Consoles (3 of 3)



```
Retail Side
JDBC Connection opened
AQ Session successfully created.
Successful getQueueTable
Successful getQueue
```

```
Result :
<ROWSET>
   <ROW NUM="1">
      <ID>526</ID>
      <ORDERDATE>2000-06-22 08:49:40.0
      <CONTACTNAME>Ora817</CONTACTNAME
      <TRACKINGNO>AX526</TRACKINGNO>
      <STATUS>Pending</STATUS>
      <CUSTOMER_ID>212</CUSTOMER_ID>
      <ITEMS>
         <ITEM_ROW NUM="1">
            <ID>2153</ID>
            <QUANTITY>1</QUANTITY>
            <ITEM_ID>4
            <ORD_ID>52
            <DISCOUNT>
         </ITEM_ROW>
      </ITEMS>
   </ROW>
</ROWSET>

Transformation done.
[Hit return to continue
```

```
   <ROW NUM="1">
      <ID>526</ID>
      <ORDERDATE>2000-06-22 08:49:40.
      <CONTACTNAME>Ora817</CONTACTNAM
      <TRACKINGNO>AX526</TRACKINGNO>
      <STATUS>Pending</STATUS>
      <CUSTOMER_ID>212</CUSTOMER_ID>
      <ITEMS>
         <ITEM_ROW NUM="1">
            <ID>2153</ID>
            <QUANTITY>1</QUANTITY>
            <ITEM_ID>410</ITEM_ID>
            <ORD_ID>526</ORD_ID>
            <DISCOUNT>0</DISCOUNT>
         </ITEM_ROW>
      </ITEMS>
   </ROW>
</ROWSET>

[Hit return to continue]
Inserting
Document processed.
```

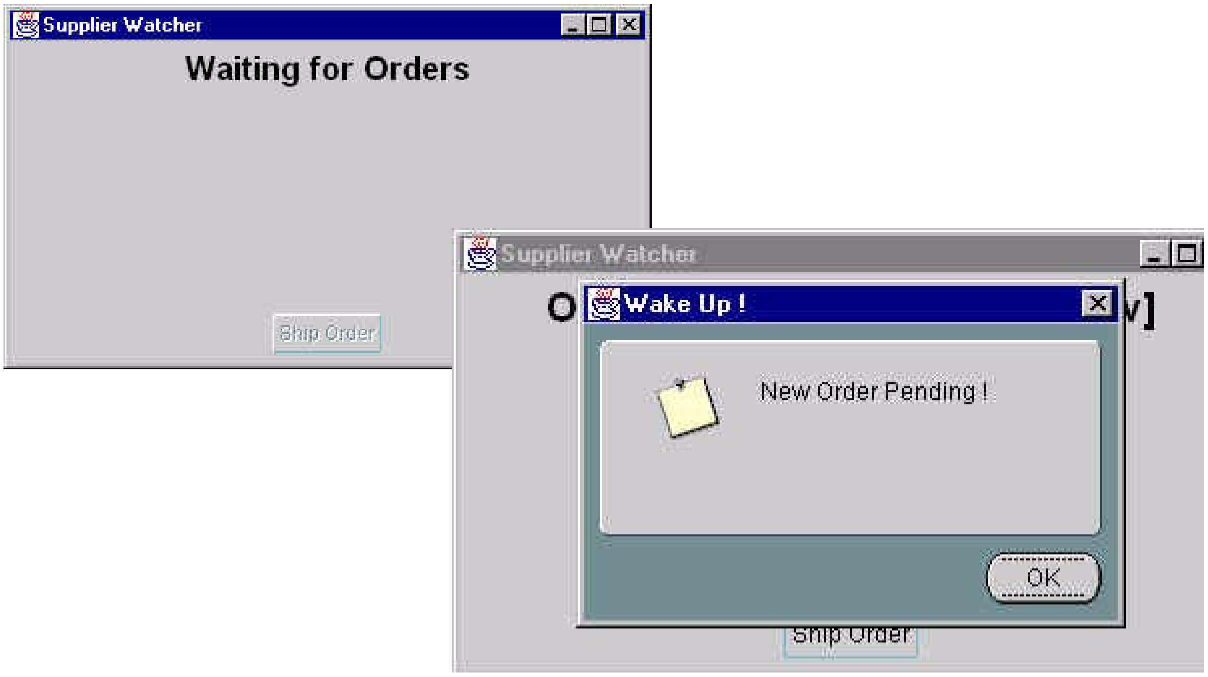
Step 6a. Supplier Application Alerts Supplier of Pending Order

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 8–21.
2. Click OK in the Supplier's Watcher dialog box. See Figure 8–22.

Scripts:

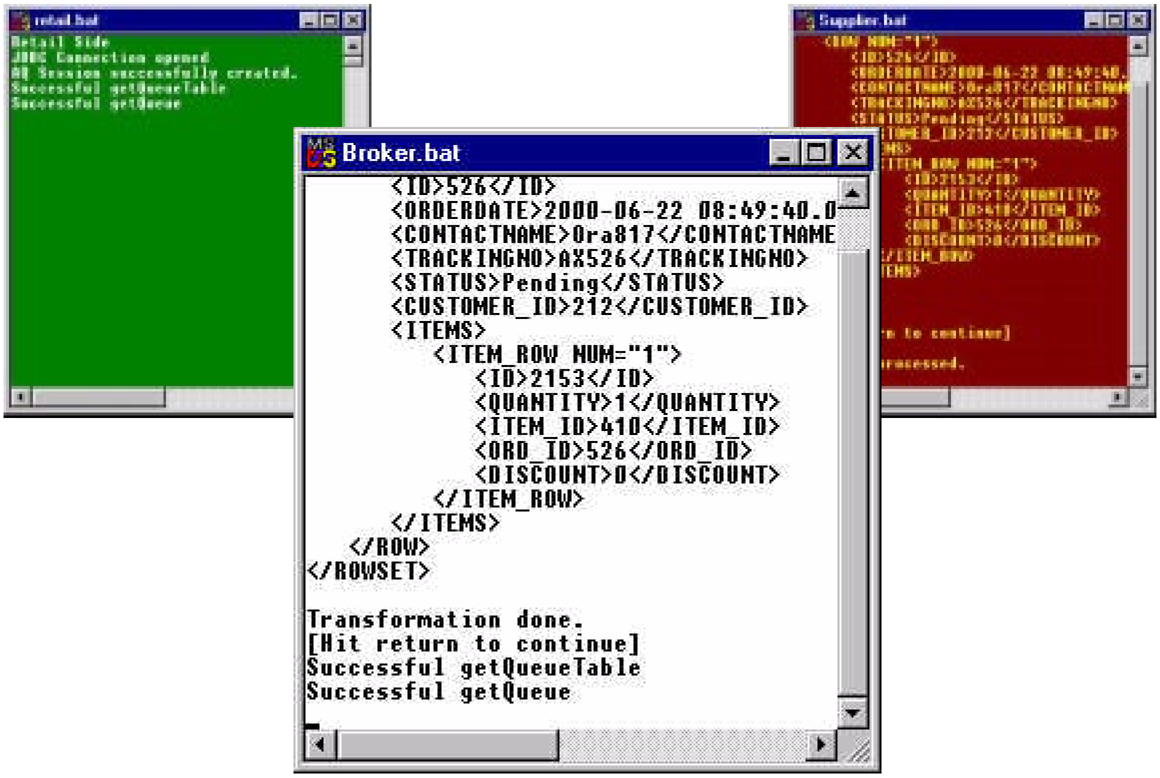
- `SupplierWatcher.java` calls `SupplierFramer.java`

Figure 8–21 The Supplier Application Alerts Supplier of Pending Order: "Wake Up!"



New Order Pending !

Figure 8–22 [Supplier]: retail.bat, broker.bat, and supplier.bat Consoles: After Clicking OK to Wake Up



```
Retail Side
JDBC Connection opened
AQ Session successfully created.
Successful getQueueTable
Successful getQueue
```

```
        <ID>526</ID>
        <ORDERDATE>2000-06-22 08:49:40.0
        <CONTACTNAME>Ora817</CONTACTNAME
        <TRACKINGNO>AX526</TRACKINGNO>
        <STATUS>Pending</STATUS>
        <CUSTOMER_ID>212</CUSTOMER_ID>
        <ITEMS>
           <ITEM_ROW NUM="1">
              <ID>2153</ID>
              <QUANTITY>1</QUANTITY>
              <ITEM_ID>410</ITEM_ID>
              <ORD_ID>526</ORD_ID>
              <DISCOUNT>0</DISCOUNT>
           </ITEM_ROW>
        </ITEMS>
     </ROW>
</ROWSET>

Transformation done.
[Hit return to contiue]
Successful getQueueTable
Successful getQueue
_
```

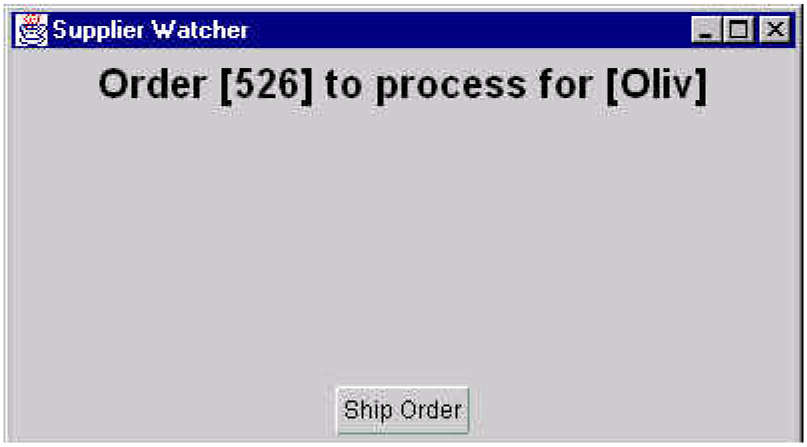
```
  <ROW NUM="1">
     <ID>526</ID>
     <ORDERDATE>2000-06-22 08:49:40.
     <CONTACTNAME>Ora817</CONTACTNAM
     <TRACKINGNO>AX526</TRACKINGNO>
     <STATUS>Pending</STATUS>
      TOMER_ID>212</CUSTOMER_ID>
      EMS>
      ITEM_ROW NUM="1">
        <ID>2153</ID>
        <QUANTITY>1</QUANTITY>
        <ITEM_ID>410</ITEM_ID>
        <ORD_ID>526</ORD_ID>
        <DISCOUNT>0</DISCOUNT>
      </ITEM_ROW>
      TEMS>


      rn to continue]

      rocessed.
```

Step 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 8–23 and Figure 8–24.

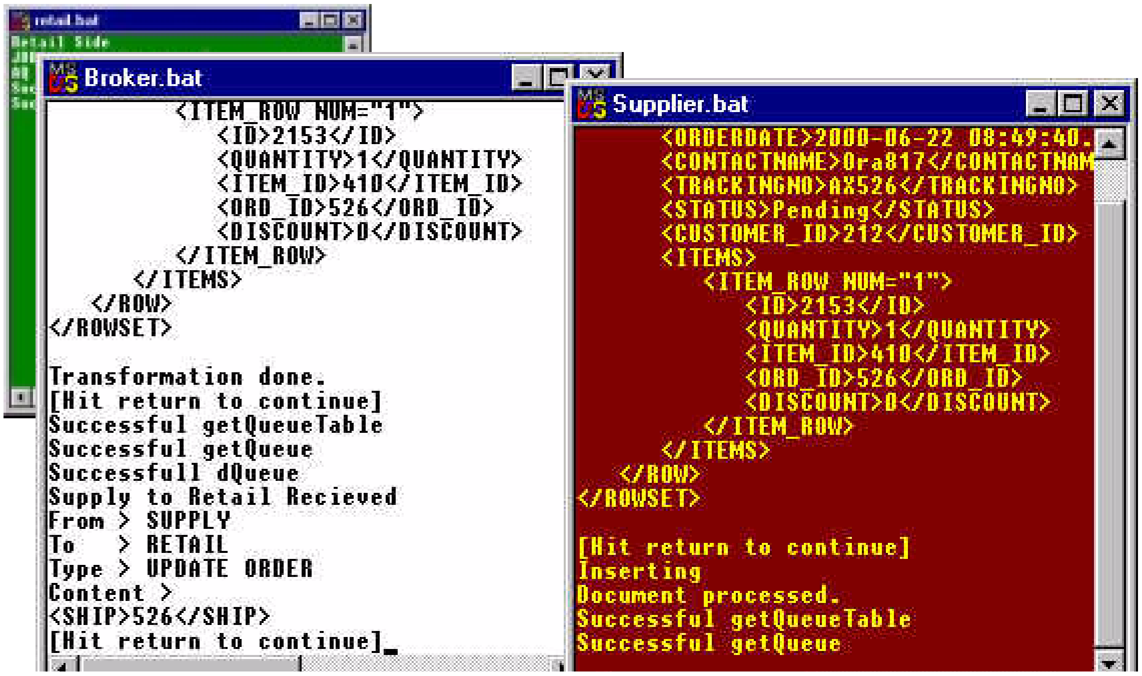
Scripts: Still using `SupplierWatcher.java`

Figure 8–23 [Supplier]: Decides to Ship the Order



Order [526] to process for [Oliv]

Figure 8–24 [Supplier]: Viewing the retailer.bat, broker.bat, and supplier.bat Consoles on "Ship Order"



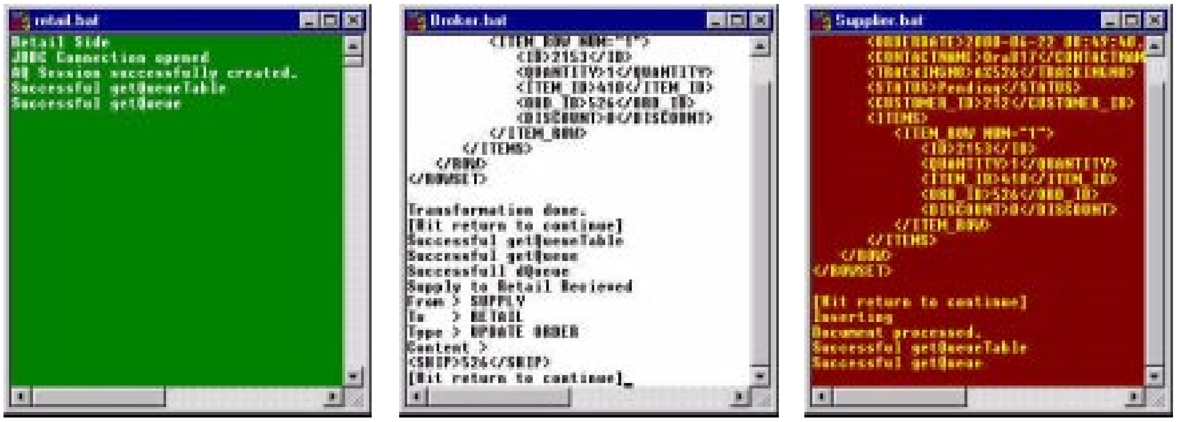
```
            <ITEM_ROW NUM="1">
               <ID>2153</ID>
               <QUANTITY>1</QUANTITY>
               <ITEM_ID>410</ITEM_ID>
               <ORD_ID>526</ORD_ID>
               <DISCOUNT>0</DISCOUNT>
            </ITEM_ROW>
         </ITEMS>
      </ROW>
</ROWSET>

Transformation done.
[Hit return to continue]
Successful getQueueTable
Successful getQueue
Successfull dQueue
Supply to Retail Recieved
From > SUPPLY
To   > RETAIL
Type > UPDATE ORDER
Content >
<SHIP>526</SHIP>
[Hit return to continue]_
```

```
      <ORDERDATE>2000-06-22 08:49:40.
      <CONTACTNAME>Ora817</CONTACTNAM
      <TRACKINGNO>AX526</TRACKINGNO>
      <STATUS>Pending</STATUS>
      <CUSTOMER_ID>212</CUSTOMER_ID>
      <ITEMS>
         <ITEM_ROW NUM="1">
            <ID>2153</ID>
            <QUANTITY>1</QUANTITY>
            <ITEM_ID>410</ITEM_ID>
            <ORD_ID>526</ORD_ID>
            <DISCOUNT>0</DISCOUNT>
         </ITEM_ROW>
      </ITEMS>
   </ROW>
</ROWSET>

[Hit return to continue]
Inserting
Document processed.
Successful getQueueTable
Successful getQueue
```

Step 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 8–25.

Figure 8–25 [Supplier]: retailer.bat, broker.bat, and supplier.bat Consoles - Form New XML Document



```
Retail Side
JDBC Connection opened
AQ Session successfully created.
Successful getQueueTable
Successful getQueue
```

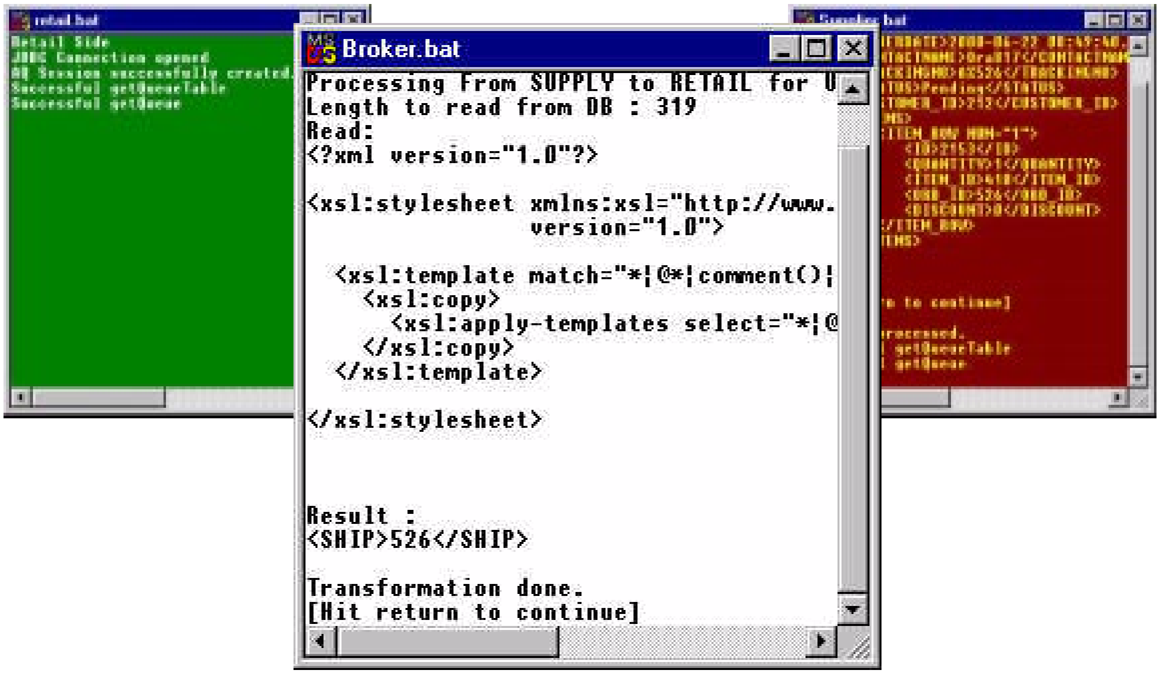
```
            <ITEM_ROW NUM="1">
               <ID>2153</ID>
               <QUANTITY>1</QUANTITY>
               <ITEM_ID>410</ITEM_ID>
               <ORD_ID>526</ORD_ID>
               <DISCOUNT>0</DISCOUNT>
            </ITEM_ROW>
         </ITEMS>
      </ROW>
</ROWSET>

Transformation done.
[Hit return to continue]
Successful getQueueTable
Successful getQueue
Successfull dQueue
Supply to Retail Recieved
From > SUPPLY
To   > RETAIL
Type > UPDATE ORDER
Content >
<SHIP>526</SHIP>
[Hit return to continue]_
```

```
         <ORDERDATE>2000-06-22 08:49:50.
         <CONTACTNAME>Ora817</CONTACTNAME
         <TRACKINGNO>AX526</TRACKINGNO>
         <STATUS>Pending</STATUS>
         <CUSTOMER_ID>212</CUSTOMER_ID>
         <ITEMS>
            <ITEM_ROW NUM="1">
               <ID>2153</ID>
               <QUANTITY>1</QUANTITY>
               <ITEM_ID>410</ITEM_ID>
               <ORD_ID>526</ORD_ID>
               <DISCOUNT>0</DISCOUNT>
            </ITEM_ROW>
         </ITEMS>
      </ROW>
</ROWSET>

[Hit return to continue]
Inserting
Document processed.
Successful getQueueTable
Successful getQueue
```

Step 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 8–26.

Figure 8–26 [AQ Broker]: retailer.bat, broker.bat, and supplier.bat Consoles - Reformat XML Document

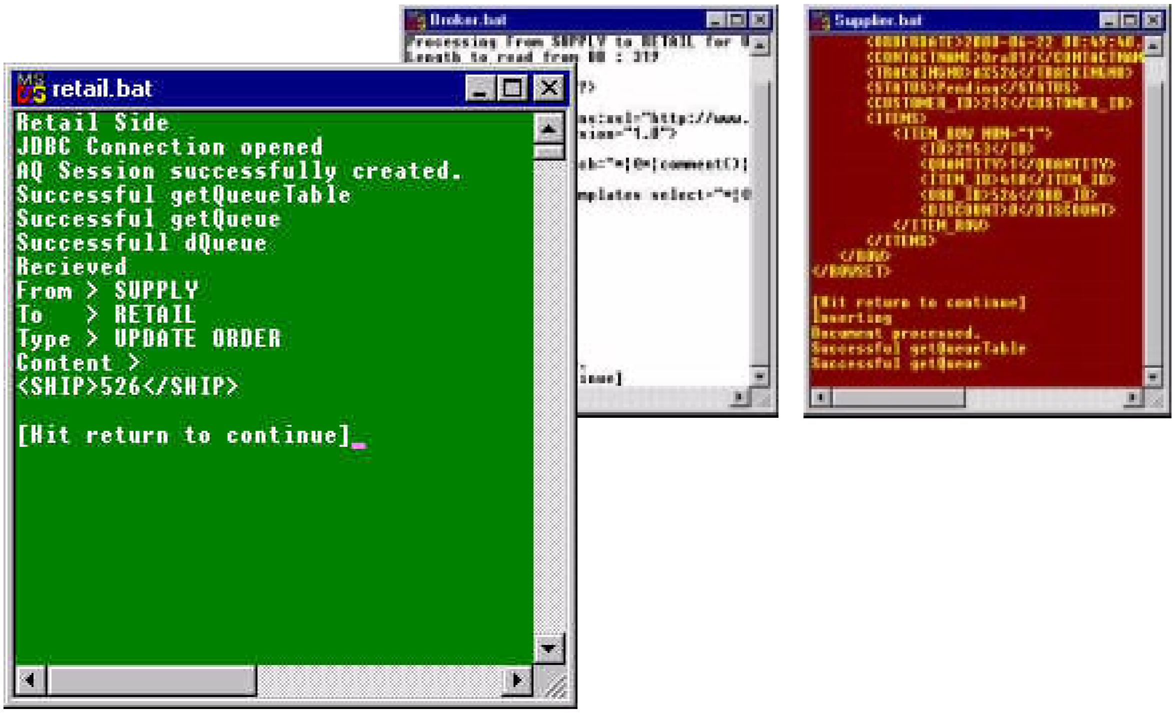


```
Retail Side
JDBC Connection opened
AQ Session successfully created.
Successful getQueueTable
Successful getQueue
```

```
Processing From SUPPLY to RETAIL for U
Length to read from DB : 319
Read:
<?xml version="1.0"?>

<xsl:stylesheet xmlns:xsl="http://www.
                version="1.0">

  <xsl:template match="*|@*|comment()|
    <xsl:copy>
      <xsl:apply-templates select="*|@
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>



Result :
<SHIP>526</SHIP>

Transformation done.
[Hit return to continue]
```

3. The document is sent to the Retailer application.
4. Hit [Return] in Retailer's Console. See Figure 8–27. This parses the XML order.

Figure 8–27 [AQ Broker]: retailer.bat, broker.bat, and supplier.bat Consoles - Sending XMLMessage



```
Retail Side
JDBC Connection opened
AQ Session successfully created.
Successful getQueueTable
Successful getQueue
Successfull dQueue
Recieved
From > SUPPLY
To   > RETAIL
Type > UPDATE ORDER
Content >
<SHIP>526</SHIP>

[Hit return to continue]_
```

```
Processing from SUPPLY to RETAIL for
Length to read from DB : 319
?>
ns:xsl="http://www.
sion="1.0">
sh:"*|@*|comment()|
mplates select="*|@
inue]
```

```
      <ORDERDATE>2000-06-22 08:49:40.
      <CONTACTNAME>Ora817</CONTACTNAME
      <TRACKINGNO>82526</TRACKINGNO>
      <STATUS>Pending</STATUS>
      <CUSTOMER_ID>212</CUSTOMER_ID>
      <ITEMS>
         <ITEM_ROW NUM="1">
            <ID>2153</ID>
            <QUANTITY>1</QUANTITY>
            <ITEM_ID>408</ITEM_ID>
            <ORD_ID>526</ORD_ID>
            <DISCOUNT>0</DISCOUNT>
         </ITEM_ROW>
      </ITEMS>
   </ROW>
</ROWSET>

[Hit return to continue]
Inserting
Document processed.
Successful getQueueTable
Successful getQueue
```

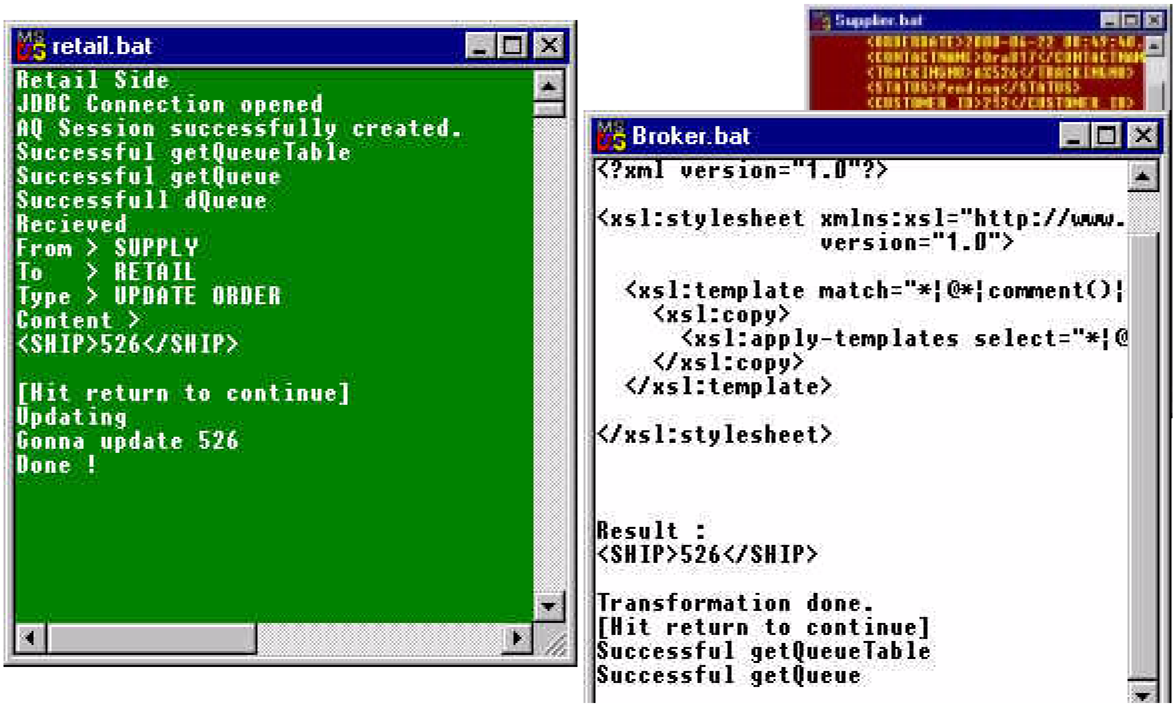
Step 8. Retailer Application Updates the Ord Table and Displays the New Order Status to Retailer

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 8–28.

Scripts:

`UpdateMaster.java`. This receives the message and parses it.

Figure 8–28 [Retailer]: retailer.bat, broker.bat, and supplier.bat Consoles - Updates Status to Shipped



```
<ORDERDATE>2000-06-22 08:49:40.
<CONTACTNAME>Ora817</CONTACTNAME
<TRACKINGNO>AS526</TRACKINGNO>
<STATUS>Pending</STATUS>
<CUSTOMER_ID>212</CUSTOMER_ID>
```

```
Retail Side
JDBC Connection opened
AQ Session successfully created.
Successful getQueueTable
Successful getQueue
Successfull dQueue
Recieved
From > SUPPLY
To   > RETAIL
Type > UPDATE ORDER
Content >
<SHIP>526</SHIP>

[Hit return to continue]
Updating
Gonna update 526
Done !
```

```
<?xml version="1.0"?>

<xsl:stylesheet xmlns:xsl="http://www.
                version="1.0">

  <xsl:template match="*|@*|comment()|
    <xsl:copy>
      <xsl:apply-templates select="*|@
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>



Result :
<SHIP>526</SHIP>

Transformation done.
[Hit return to continue]
Successful getQueueTable
Successful getQueue
```

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
- Java Example 7: ParserTest.java
- Java Example 8: TableInDocument.java
- Java Example 9: XMLFrame.java
- Java Example 10: XMLProducer.java
- Java Example 11: XMLtoDMLv2.java
- Java Example 12: XMLGen.java
- Java Example 13: XMLUtil.java
- Java Example 14: XSLTWrapper.java

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();
    }
  }
```

9

Service Delivery Platform (SDP) and XML

This chapter contains the following sections:

Oracle Service Delivery Platform

This chapter introduces you to the Oracle Service Delivery Platform (SDP).

SDP Business Solutions

Many solutions using Business-to-Business XML message payloads can be built on the Service Delivery Platform (SDP). These include the following:

- Phone Number Portability
- Wireless Number Portability and Data Service Automation
- NPAC
- Service Gateway
- ADSL
- Voice over IP

Other possible solutions include Bandwidth Exchange and Carrier Pre-Selection.

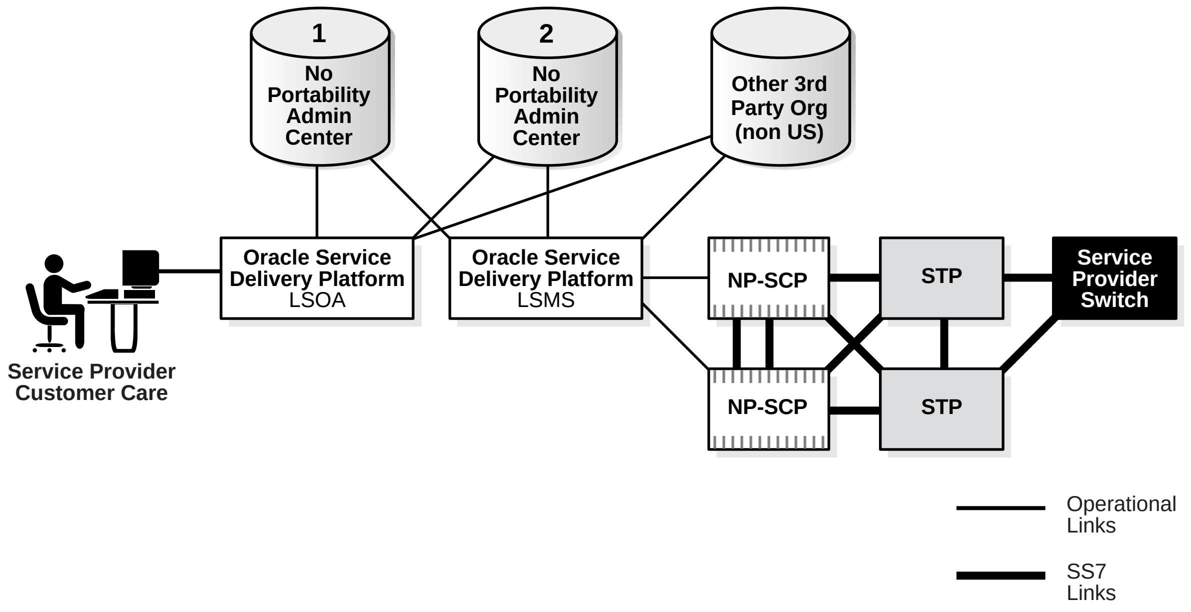
Phone Number Portability

Phone 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.

Number Portability application 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.

Figure 9–1 Number Portability and Related Architecture


1
No Portability Admin Center
2
No Portability Admin Center
Other 3rd Party Org (non US)
Oracle Service Delivery Platform LSOA
Oracle Service Delivery Platform LSMS
Service Provider Customer Care
NP-SCP
STP
Service Provider Switch
NP-SCP
STP
Operational Links
SS7 Links

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) as shown in Figure 9–2.

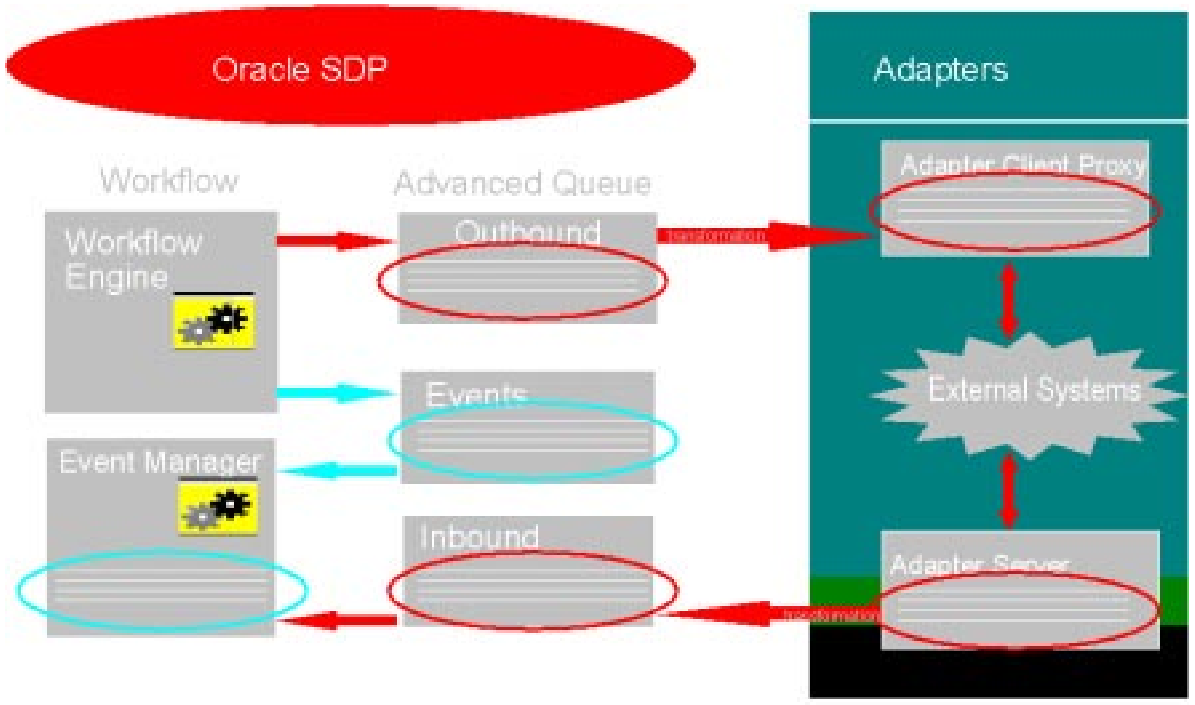
The "Message Builder" module creates and enqueues the XML messages. The Communication Protocol Adapter ("Adapter") starts dequeueing the messages and sends them to the external systems.

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.

Figure 9–2 Messaging Using Advanced Queuing



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.

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
- Assign different nodes of the XML message to an Oracle data source, an SQL query or a stored procedure.
- Nest SQL queries

For example, you can get a list of depts and all emps in each dept in an XML message by performing 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 that "listen" 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 to the method, `performControlMessageProcessing.` This allows for a dynamic number of parameters to be passed to the adapter.

For example, to start an adapter 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 terms are used in this chapter:

- GSM — Global System for Mobile Communication. This is an internationally accepted digital cellular telephony standard which requires frequencies in the 900M band to allow for roaming.
- NPAC — Number Portability Administration Center
- NRC — Number Registration Center. Another name for NPAC.
- SDP — Oracle Service Delivery Platform (SDP)

Wireless Number Portability (WNP)

The Inter-Carrier Communication process in a Wireless Number Portability (WNP) solution must take in to consideration, communication between the following:

- Wireless and Wireless Providers
- Wireless and Wireline providers.

Communication between wireless providers can be further divided according to the following carrier type porting:

- Porting between GSM carriers
- Porting between cellular carriers
- Porting between GSM and cellular carriers

Mobile Directory Number (MDN) and Mobile Subscriber Identity (MSID)

In a wireless system, a subscriber can be identified by the Mobile Directory Number (MDN) and Mobile Subscriber Identity (MSID). The MDN is portable and dialable. MSID is not-portable and not-dialable, and is used to identify the home service provider of a roamer. The MSID is also re-usable.

In the GSM world (in North America) the MDN is a 10-digit telephone number and is referred to as the MSISDN (Mobile Subscriber ISDN). The MSID of a GSM carrier is a 1-15 digit number referred to as the IMSI (International Mobile Subscriber Identity).

In the cellular world the MDN and MSID (referred to as MIN - Mobile Identification Number) are both 10-digit directory numbers (in the past both numbers have been the same).

A WNP solution must be able to support all these types of porting requests.

See Also: The Porting Process - Reference CTIA Numbering Advisory Working Group Report On Wireless Intercarrier Communications Version 2.0.

NPAC

In many countries where number portability is mandated, an organization such as Number Porting Administration Center (NPAC) controls the cut over of service from one operator to another. It also records information about ported numbers, and provides a centralized point from which porting information can be distributed.

Having a centralized and impartial body to oversee the porting process provides the following benefits:

- There is a single national record of ported number. This database facilitates access to all information regarding ported numbers for a new network operator. The database also allows synchronization function in case any single operator's internal database became corrupted or out of sync (disaster recovery).
- It relieves the recipient or donor to inform all other operators of the ported number (only the last two models provide this).

In addition to the above, a centralized database can be a reference for civic service organizations, such as:

- Directory Service Providers
- Emergency Services,...

Service Gateway

When offering a service to subscribers involves two Telecommunications Operators (TO), or Service Providers (SP), they must be able to interconnect their separate OSSs. This would require a number of policies about:

- The extent to which business processes of one partner influence those of another and
- The extent to which one partner can autonomously change information held by the other.

At the heart of this set of requirements is the need for co-operating organizations to control the integrity of their data.

Service gateway offers flow-through functionality for inter-enterprise processes while enabling:

- Seamless Service: While the service is provided by cooperation of more than one provider, it needs to seem seamless to the end customer.
- Autonomy: Even though the service providers cooperate in providing a service, they must maintain their autonomy.
- Simplicity: Configuring new services, tracking down services, and making changes to the business process must not be complex.
- Technology Choice: The cooperating entities must be allowed to choose any technology based on their own criteria. So the service gateway must be technology agnostic.

Standards Adherence: The gateway must follow common practices and standards whether set by regulatory organizations, industry forums and standard bodies, or by market conditions.

Asymmetric Digital Subscriber Line (ADSL)

ADSL is an Asymmetric Digital Subscriber Line.

ADSL Business Procedure

A typical ADSL business process occurs as follows:

1. Pre-qualify check
2. Pre-order feasibility check
3. Create Order

4. Perform loop qualification check
5. Provision DSL line
6. Verify DSL loop
7. Deliver DSL CPE
8. Schedule and Install DSL CPE
9. Test DSL CPE with DSL and ATM connection
10. Test IP service with DSL line
11. Publish order completion message

ADSL Benefits

The following are ADSL benefits

- Time to market: Since DSL solutions provide predefined / preconfigured business flows, data, configuration, and interfaces, customers can implement or modify DSL services fast.
- Benefits to the customer are large as DSL services are gaining momentum in the communications space.

DSL in a Nutshell

The DSL technology is a new technology platform that is leading customers into the future. DSL is a broadband digital connection made directly to the customer premises using existing copper telephone lines. With DSL technology, businesses can take advantage of a large suite of services designed to enhance the use of the Internet and the productivity of its users. The DSL network components consist of a multi services DSLAM located at the CO and a DSL remote transceiver unit (DSL modem) located at the customer premises. The DSLAM provides backhaul services for packet, cell and / or circuit-based applications through the concentration of the DSL lines onto T1 / E1, T3 / E3 or ATM outputs. The DSL modem maintains a digital link from customer business to the network. This modem can drive ordinary telephone lines at speeds far beyond conventional dial-up modems (up to 7+ Mbps). This modem plugs into existing local area network or can be attached to a PC using a special cable.

There are many varieties of DSL technology, collectively called xDSL:

- ADSL - Asymmetric Digital Subscriber Line - This technology reports a downstream speed, but its upstream speed is a fraction of the downstream. Primarily used in residential applications and many providers do not guarantee its bandwidth levels. ADSL supplies three separate frequency channels over the same phone line. Phone conversations are carried on one channel, data from a Network Service Provider (NSP) to the end-user is transferred downstream on one channel, and the third channel carries data upstream from the end-user to the Network Service Provider.

- SDSL - Symmetric Digital Subscriber Line - This technology provides the same bandwidth in both directions, upstream and downstream. That means whether you're uploading or downloading information, you have the same high-quality performance. SDSL provides transmission speeds within a T1/E1 range, of up to 1.5 Mbps at a maximum range of 12,000 - 18,000 feet from a central office, over a single-pair copper wire. This option is ideal for small- and medium-sized businesses that have an equal need to download and upload data over the Internet.
- RADSL - Rate Adaptive Digital Subscriber Line - This technology automatically adjusts the access speed based upon the condition of the line.
- IDSL - ISDN Digital Subscriber Line - This technology is symmetrical, similar to SDSL, but operates at slower speeds and longer distances.
- HDSL - High-Data-Rate Digital Subscriber Line - This technology is symmetrical, but is mainly deployed for PBX over a T-1 line.
- VDSL- Very-High-Rate Digital Subscriber Line - This is a high-speed technology, but has a very limited range.

Using DSL technology, service providers can provide a series of end-to-end, business managed solutions to give customer the leading edge with Core Internet Services, Web Hosting, Enhanced email, managed equipment, Virtual Private Networks and Server Based Applications. This should help customer to operate better and faster and maximizes the use of customer resources.

Examples of the DSL-based services can be provided are:

- Web hosting -manage the day-to-day technical administration of customer Web site so customer can concentrate on core business.
- Enhanced email - access email anytime, anywhere. Enables sending/receiving through common email clients. Email accessibility via the web from any computer with Internet connectivity.
- High-speed access to Internet-based video
- Remote local area network access or corporate network access

- DNS Management - ensure proper handling of customer Web identity and management of customer system.
- Home shopping

Voice Over IP (Clarent)

The Clarent VoIP system consists of the following components:

- *Command Centers (CCC)* that interface with Call Managers (CM) Gateways (GW)
- *Customer/Provisioning Database*, CM that interface with CPE
- Gateway that interfaces with PSTNs, IP networks, and other gateways

The Customer / Provisioning Database contains Customer, Routing (Trunks), Rating, and other tables, that the CCC uses to activate a customer.

For example, suppose Service Provider X, uses the Clarent Network to provide VoIP service to its customers. If a customer requests VoIP service, then X must enter all the information required to activate the customer in Clarent's Customer / Provisioning database. This information will be used by the CC to activate VoIP service for that customer.

Currently customer / route / rating data is added, one-by-one, via Clarent Assist.

Clarent Assist is an HTML screen which can be used to add, modify and delete rows in the Clarent Customer / Provisioning database.

Bandwidth Exchange (Prototype)

The Communications Services Exchange Prototype (called "The Exchange" within herein) is intended to be a Telco solution using the SDP engine. The Exchange is an Internet Community that will be used to sell, purchase and facilitate wholesale communications services between peer Service Providers as well as large Companies & Corporations. The Exchange only facilitates transactions between Buyers and Sellers; Both Buyers and Sellers can be large corporations who buy or sell services from each other or from Service Providers other than their own; the corporations still belong to their incumbent Service Providers even after they exchange the services with other Service Providers. Once Buyers and Sellers agree and close the transaction, options are available for them to choose to provision and activate the exchanged services through a Service Fulfillment capability provided by The Exchange.

The Exchange functionality and platform will be based on Oracle e-Business Internet Procurement application (refer to “Oracle Exchange”). Any additional functionality required will be developed by the SDP Telco Solution. The major components and functions of The Exchange are comprise of the following:

- Registration and Profiling Management - based on Oracle Exchange. Major functions include: marketplace administrator reviews and approves participants, security rules at the business function and data access level, users manage their own personal profiles and preferences)
- Products & Services Catalog - based on Oracle Exchange and CRM Product Catalog. Major functions include: self-service tools, XML open interfaces, create and maintain catalog.
- Service Level Agreement (SLA) - based on Oracle Exchange. Major functions include: Create and distribute RFIs and RFQs (Bids) with SLA on Service Bulletins, Maintain standard template for SLAs, Bid analysis and award, single of multiple rounds of bidding, download / upload Service Bulletins, event driven notifications
- Security - based on Oracle Exchange. Major functions: end user / company data security, connectivity
- Buyer and Seller Self-Service - based on Oracle Exchange. Major functions: HTML based inquiry and transactional capability, review and accept new orders, enable every buyer and seller or member to participate in the marketplace.
- Service Configurator - new for The Exchange. Major functions: automatically populate or an input capability (may be a file) for the seller to provide service configuration data, maintain (view, add, change and delete) service configurator,
- Service fulfillment - new for The Exchange. Major functions: convert Oracle Exchange order to provisionable service order accepted by SDP, use SDP fulfillment engine to provision and activate or remove network elements based on the service configuration,

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

Figure 9–3 and Figure 9–4 illustrate the Number Portability overall workflow.

Figure 9–3 Number Portability Workflow (1 of 2)

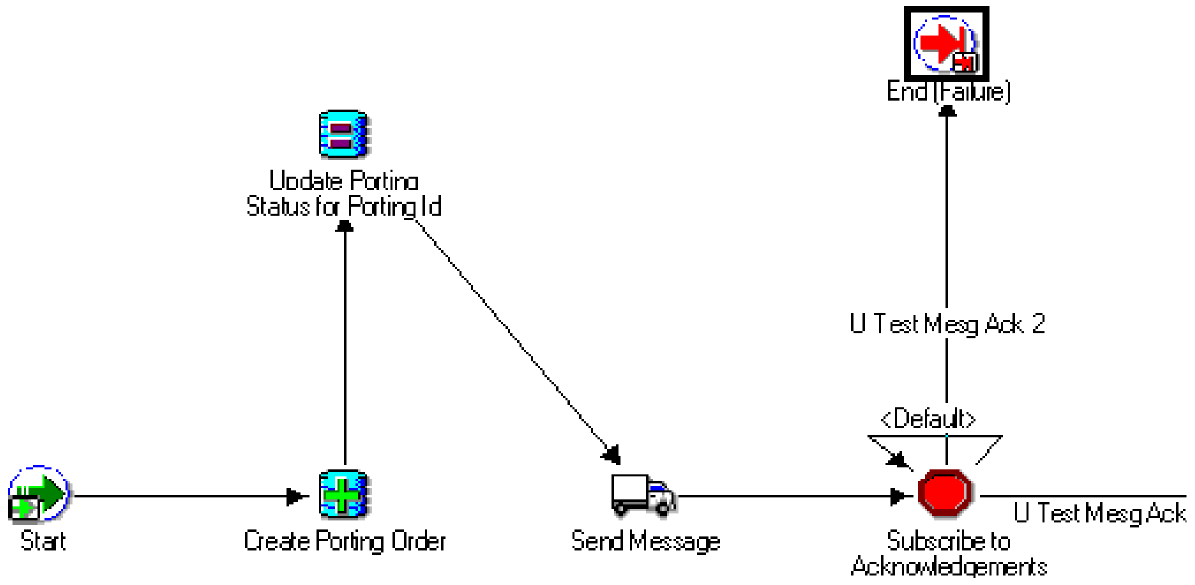
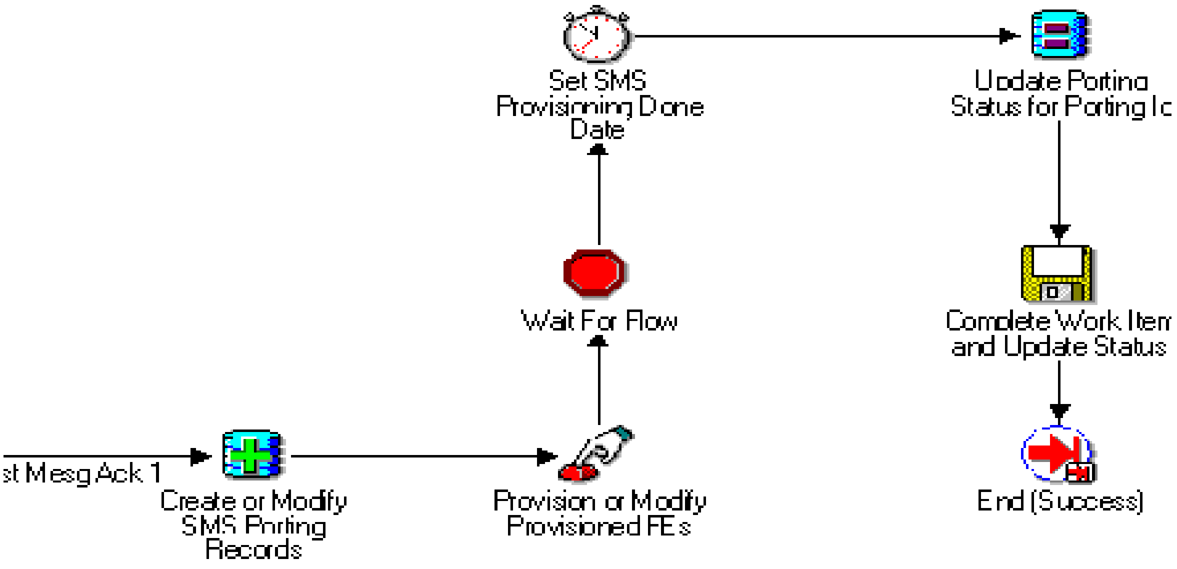


Figure 9–4 Number Portability Workflow (2 of 2)


st Mesg Ack 1
Create or Modify SMS Porting Records
Provision or Modify Provisioned FEs
Wait For Flow
Set SMS Provisioning Done Date
Update Porting Status for Porting Ic
Complete Work Item and Update Status
End (Success)

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 Engine

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.

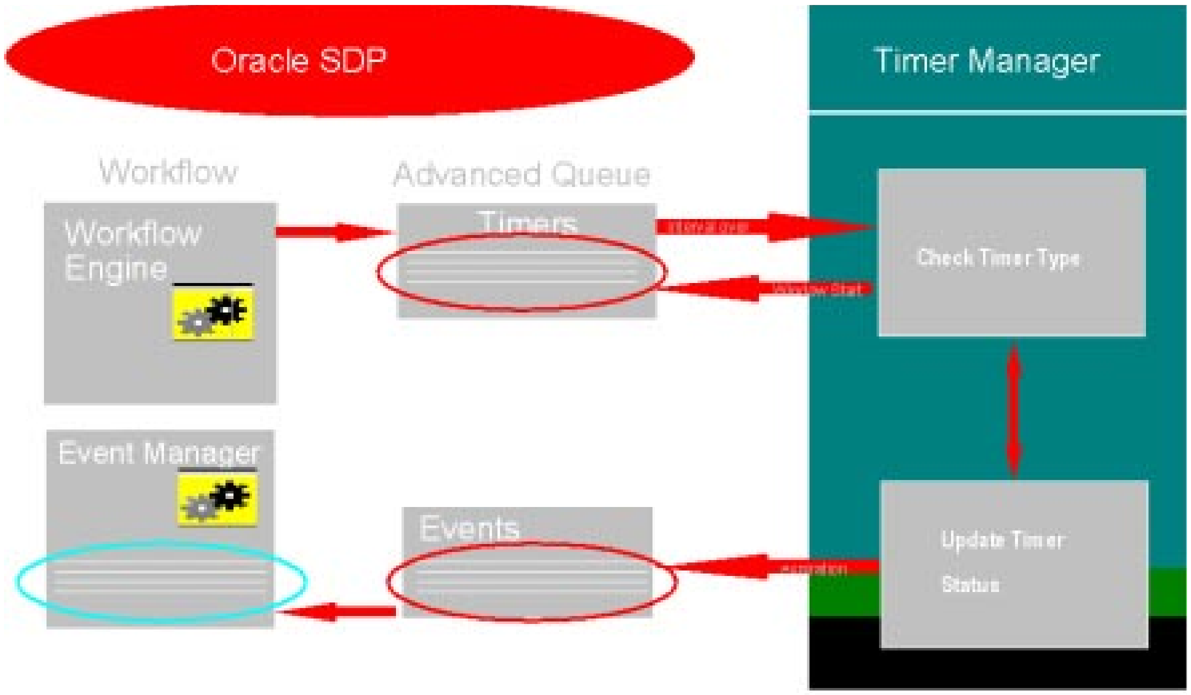
The Fulfilment Engine has the following main features:

- Allows for Network Element Provisioning
- Tight Integration with Oracle Provisioning
- Provisioning Protocol Adapters
  - Interactive (e.g. Telnet)
  - Script (any executable e.g. PERL)
  - HTTP

- Standard Activities for NP SMS Functionality
    - Support for Served Number Ranges Setup

See Figure 9–5.

Figure 9–5 Oracle Provisioning Integration in Number Portability (NP)



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,...

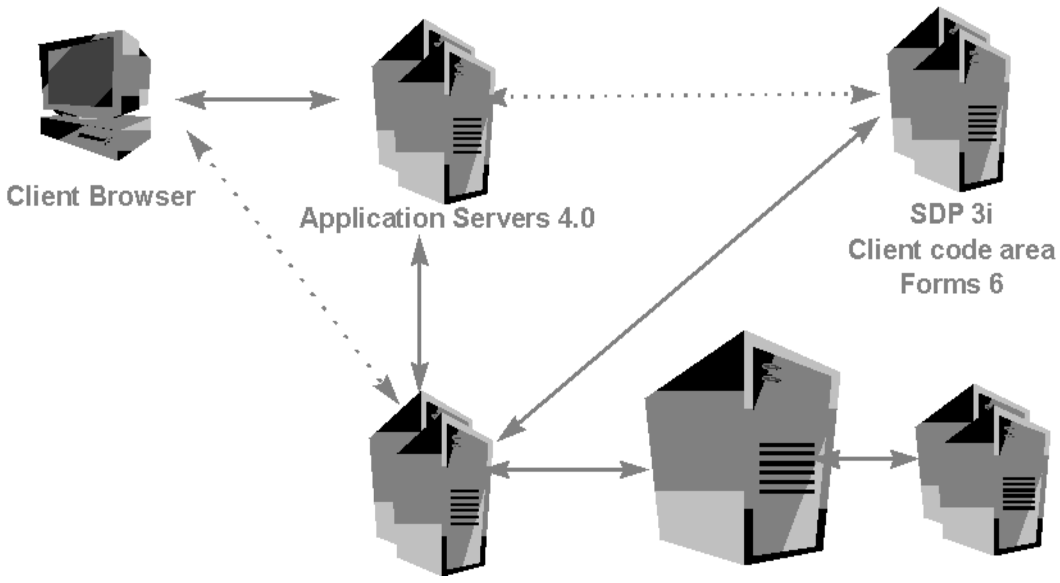
Requirements for Building a Phone Number Portability Application

To build a Number Portability application, you need the following:

- Oracle Applications 11*i*
- Oracle SDP Number Portability 11i Release 2
- Oracle8i Release 8.1.5 Enterprise Edition or above with the Objects Option
- Oracle8 Release 8.0.5 / 8.1.5 and above JDBC Drivers
- Oracle Developer Release 6.0.5 Runtime or above
- Oracle Workflow 2.5.0 Cartridge (Standalone Version!)
- Oracle9*i* Application Server 1.0.1 and above
- JDK / JRE 1.1.7 and above
- JDK 1.1 and above Enabled Browser
- (Netscape 4.5+ / MS IE 5.0 recommended for XML)

Figure 9–6 shows the Number Portability environment. The servers on the bottom of the figure are from left to right, a Forms Server, Oracle9i Application Server, and the SDP Adapter Server. This is just schematic. The servers could reside on one machine.

Figure 9–6 Number Portability Environment



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 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.

Message Builder (*i*Message) Features

The SDP Message Builder (*i*Message) has the following features:

- GUI based message definition using iMessage Studio
    - Support for Oracle XML Parser/Utilities in future release
    - Messages and Events pre-defined at site configuration
- Supports messages, events and timers
- Uses Internet Standard XML Message Format
    - Works with XML “Dekagrams” for all messages
- Automatic Optimized Code Generation
    - Minimal Coding for Processing and Validation Logic
    - Specify message data source in SQL, PL/SQL Functions or SDP Order / Work Item / FA Parameters declaratively
    - Generates Send(), Publish(), Process(), Validate(), Fire() based on Message Type
- Planned integration with “Oracle XML Gateway” and Oracle Integration Server (OIS)

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 9–7, "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.

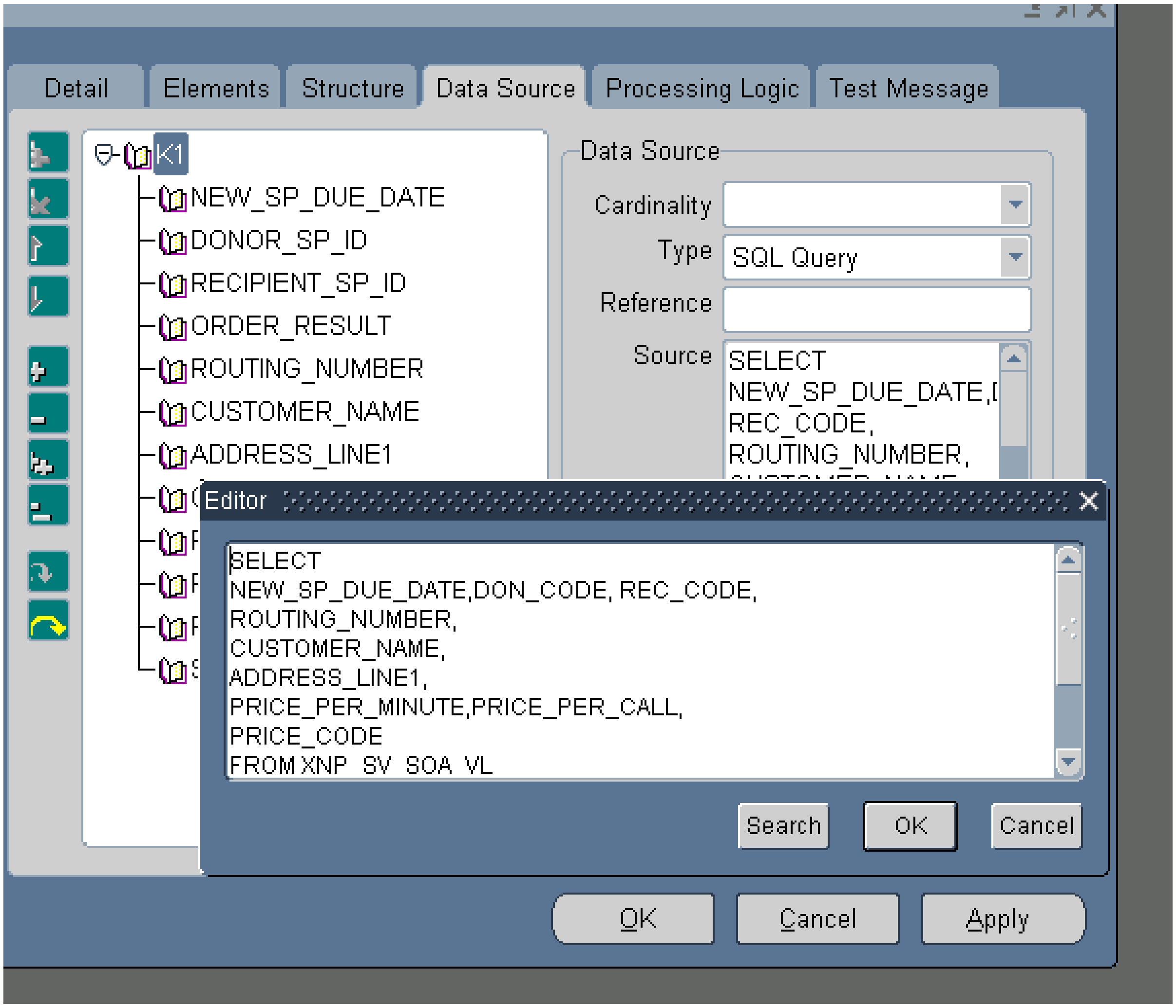
A number of steps are involved when using *i*Message to define your XML message sets. These include the following:

- *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 9–7.

Figure 9–7 Using iMessage's Data Source Window to Define the Data Source for XML Message Elements (in Oracle Developer Forms)



Type: SQL Query

```
SELECT
NEW_SP_DUE_DATE,DON_CODE, REC_CODE,
ROUTING_NUMBER,
CUSTOMER_NAME,
ADDRESS_LINE1,
PRICE_PER_MINUTE,PRICE_PER_CALL,
PRICE_CODE
FROM XNP_SV_SOA_VL
```

Supported SDP datatypes include the following:

- *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. Columns in the SQL query can be used as a reference for other message elements provided these message elements are defined at a lower level in the tree hierarchy.

Other data types include the following:

- SDP Order Parameters
- SADP Work Item Parameters
- SDP Fulfilment Action Parameters

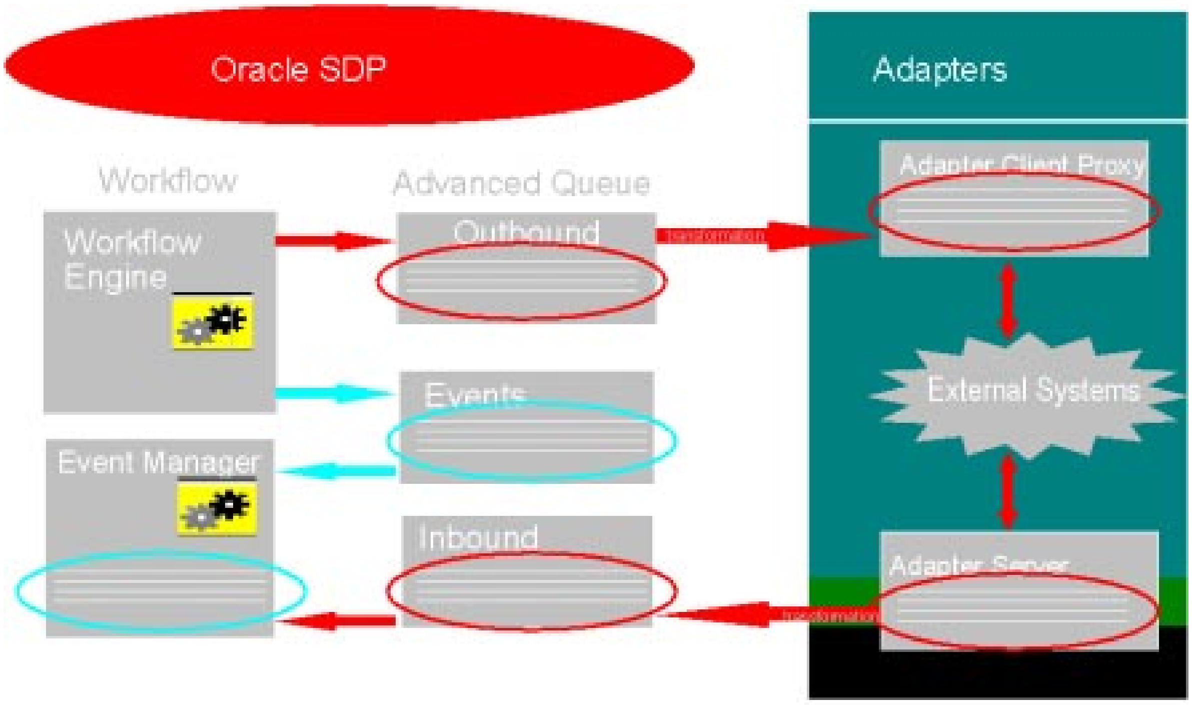
Using Timer Manager

The Timer Manager has the following main features:

- Uses XML Message Builder to define Timers
- Supports Multiple Types of Timers
- Process Timers (explicit Fire and Remove)
- Window Timers (implicit Fire after a "Delay")
- Message Timers (fire timers related to a message)
- Easy integration with external SLA systems
- API hooks to determine Delay and Interval
- Timer API and Standard Workflow Activities
- Fire(), Remove(), CheckTimerStatus(), Restart(), Recalculate(), StartRelatedTimers() etc.
- Useful for Jeapordy Management
- Define Jeapordy Events as Timers
- Send Notifications or start workflows on timer expiration

Figure 9–8 shows how SDP's Timers implement Advanced Queueing.?????

Figure 9–8   Timers Using Advanced Queueing



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.

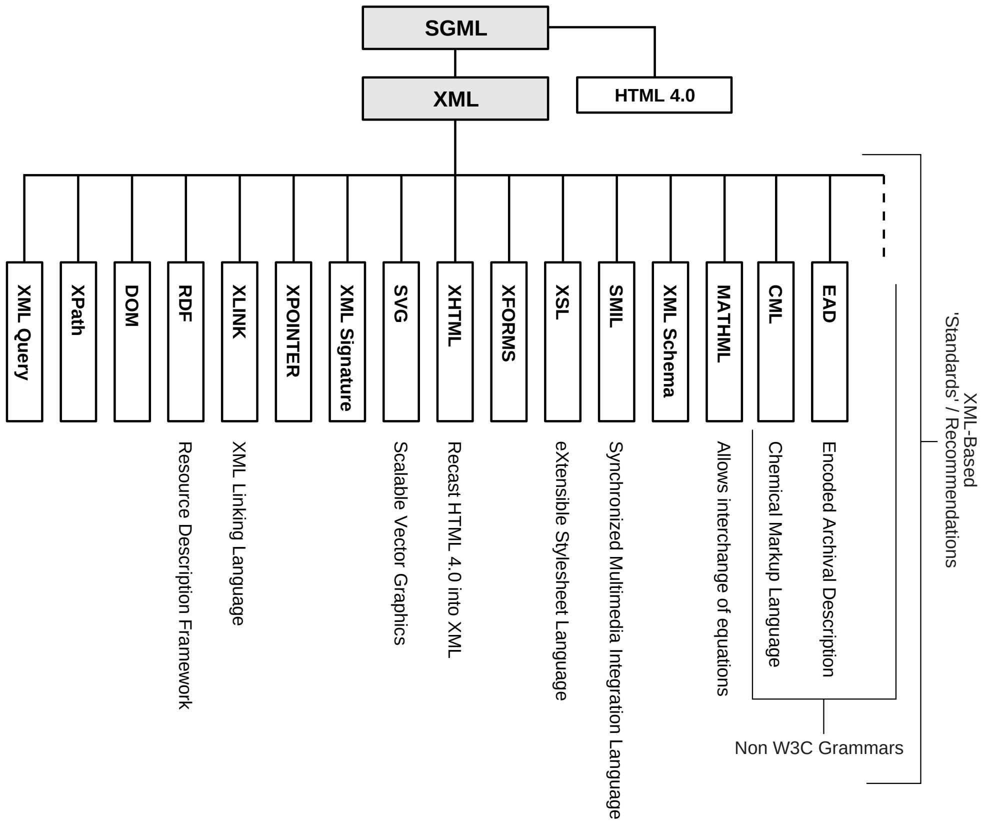
- *XML 1.0* was recommended by W3C in February 1998. It has resulted numerous additional W3C Working Groups, a Java Platform Extension Expert Group, and the XML conversion of numerous data interchange standards such as Electronic Data Interchange (EDI). The next version of HTML will be an XML application 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 definition XML document.
- *XSL*. XSL consists of two W3C recommendations:
  - XSL Transformations for transforming one XML document into another
  - XSL Formatting Objects for specifying the presentation of an XML document
- *XPath*. XPath is the W3C recommendation that specifies the data model and grammar for navigating an XML document utilized by XSL-T, XLink, and XML Query.
- *XPointer*. XPointer is the W3C recommendation that specifies the identification of individual entities or fragments within an XML document using XPath

navigation. This W3C proposed recommendation is defined at http://www.w3.org/TR/WD-xptr

- *DOM*. The W3C recommendation that specifies the Document Object Model of an XML Document including APIs for programmatic access.

The XML family of applications is illustrated in Figure A–1.

Figure A–1 The XML Family of Applications ('Including XML-Based Standards')



XML Features

The following bullets describe XML features:

- **Data Exchange, From Structured to Unstructured Data**: XML enables a universal standard syntax for exchanging data. XML specifies a rigorous, text-based way to represent the structure inherent in data so that it can be authored and interpreted unambiguously. Its simple, tag-based approach leverages developers' familiarity of HTML but provides a flexible, extensible mechanism that can handle the gamut of "digital assets" from highly structured database records to unstructured documents and everything in between. " W3C
- **SGML Was Designed Specifically for Documents - XML is Designed for Potentially Any Data**: The SGML markup language was specifically designed for documents. Web-centric XML is like a toolkit that can be used to write other languages. It is not designed for documents only. Any data that can be described in a tree can be programed in XML.
- **A Class of Data Objects - A Restricted Form of SGML**: www.oasis-open.org describes XML as follows: "... XML, describes a class of data objects called XML *documents* and partially describes the behavior of computer programs which process them. XML is an application profile or restricted form of SGML, the Standard Generalized Markup Language. By construction, XML documents are conforming SGML documents."
- **XML's Many Uses...**: A W3C.org press release describes XML as follows: "... XML is primarily intended to meet the requirements of large-scale Web content providers for industry-specific markup, vendor-neutral data exchange, media-independent publishing, one-on-one marketing, workflow management in collaborative authoring environments, and the processing of Web documents by intelligent clients.
- **Metadata**. XML is also finding use in certain metadata applications.
- **Internationalization**. "XML is fully internationalized for both European and Asian languages, with all conforming processors required to support the Unicode character set in both its UTF-8 and UTF-16 encoding..." Its primary use is for electronic publishing and data interchange..."
- **Parsed or Unparsed Storage Entities:** From the W3C.org XML specification proposal: "... XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form the character data in the document, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure.

- **XML Processor Reads XML Documents**. "... XML provides a mechanism to impose constraints on the storage layout and logical structure. A software module called an XML processor is used to read XML documents and provide access to their content and structure. It is assumed that an XML processor is doing its work on behalf of another module, called the application...."
- **Open Internet Standard.** XML is gaining wide industry support from other vendors besides, like IBM, Sun, Microsoft, Netscape, SAP, CISCO and others, as a platform- and application-neutral format for exchanging information.

Although this manual is not intended to expound on XML syntax, a brief overview of some key XML topics is presented here. You can refer to the many excellent resources listed in "Additional XML Resources" for more information on XML syntax.

How XML Differs From HTML

Like HTML, XML is a subset of SGML (Structured Generalized Markup Language), optimized for delivery over the web.

Unlike HTML, which tags elements in web pages for presentation by a browser, 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 9–1 summarizes, how XML differs from HTML.

Table 9–1 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 in Oracle9i Application Developer's Guide - XML:

- "Using XSL and XSLT"
- "Using XML Parser for Java"

In this manual, see Chapter 7, "Customizing Discoverer 4i Viewer with XSL".

eXtensible Stylesheet Language (XSL)

eXtensible Stylesheet Language (XSL), the stylesheet language of XML is another W3C recommendation. XSL provides for stylesheets that allow you to do the following:

- Transform XML into XML or other text-based formats such as HTML
- Rearrange or filter data

- Convert XML data to XML that conforms with another Document Type Definition (DTD), an important capability for allowing different applications to share data

Cascading Style Sheets (CSS)

Cascading Style Sheets (CSS1), a W3C specification was originally created for use with HTML documents. With CSS you can control the following aspects of your document's appearance:

- Spacing. Element visibility, position, and size
- Colors and background
- Fonts and text

CSS2 was published by W3C in 1998 and includes the following additional features:

- System fonts and colors
- Automatic numbering
- Supports paged media
- Tables and aural stylesheets

'Cascading' here implies that you can apply several stylesheets to any one document. On a web page deploying CSS, for example, three stylesheets can apply or cascade:

1. User's preferred stylesheet takes precedence
2. Cascading stylesheet
3. Browser stylesheet

> **See Also:** Oracle9i Application Developer's Guide - XML, "Using XSL and XSLT"

Extensibility and Document Type Definitions (DTD)

Another key advantage of XML over HTML is that it leaves the specification of the tags and how they can be used to the user. You construct an XML document by creating your own tags to represent the meaning and structure of your data.

Tags may be defined by using them in an XML document or they may be formally defined in a Document Type Definition (DTD). As your data or application requirements change, you can change or add tags to reflect new data contexts or extend existing ones.

The following is a simple DTD for the previous XML example:

```
<!ELEMENT EMPLIST (EMP)*>
<!ELEMENT EMP (EMPNO, ENAME, JOB, SAL)>
<!ELEMENT EMPNO (#PCDATA)>
<!ELEMENT ENAME (#PCDATA)>
<!ELEMENT JOB (#PCDATA)>
<!ELEMENT SAL (#PCDATA)>
```

Note: The DOCTYPE declaration is only used when the DTD is embedded in XML code.

Well-Formed and Valid XML Documents

Well-Formed XML Documents

An XML document that conforms to the structural and notational rules of XML is considered *well-formed*. A well-formed XML document does not have to contain or reference a DTD, but rather can implicitly define its data elements and their relationships. Well-formed XML documents must follow these rules:

- Document must start with the XML declaration, <?xml version="1.0">
- All elements must be contained within one root element
- All elements must be nested in a tree structure without overlapping
- All non-empty elements must have start and end tags

Valid XML Documents

Well-formed XML documents that *also* conform to a DTD are considered *valid*. When an XML document containing or referencing a DTD is parsed, the parsing application can verify that the XML conforms to the DTD and is therefore valid, which allows the parsing application to process it with the assurance that all data elements and their content follow rules defined in the DTD.

Why Use XML?

XML, the internet standard for information exchange is useful for the following reasons:

- *Solves Data Interchange Problems*. It facilitates efficient data communication where the data:
  - Is in many different formats and platforms
  - It must be sent to different platforms
  - Must appear in different formats and presentations
  - Must appear on many different end devices

  In short, XML solves application *data interchange* problems. Businesses can now easily communicate with other businesses and workflow components using XML. See Chapters 2 through 20 for more information and examples of how XML solves data interchange problems.

  Web-based applications can be built using XML which helps the interoperation of web, database, networking, and middleware. XML provides a structured format for data transmission.
- *Industry-Specific Data Objects are Being Designed Using XML*. Organizations such as OAG and XML.org are using XML to standardize data objects on a per-industry basis. This will further facilitate business-to-business data interchange.
- *Database-Resident Data is Easily Accessed, Converted, and Stored Using XML*. Large amounts of business data resides in relational and object-relational tables as the database provides excellent data queriability, scalability and availability. This data can be converted from XML format and stored in object-relational and pure relational database structures or generated from them back to XML for further processing.

Other Advantages of Using XML

Other advantages of using XML include the following:

- You can make your own tags
- Many tools support XML
- XML is an Open standard

- XML parsers built according to the Open standard are interoperable parsers and avoid vendor lock-in. XML specifications are widely industry approved.
- In XML the presentation of data is separate from the data's structure and content. It is simple to customize the data's presentation. See "Presenting XML Using Stylesheets" and "Customizing Your Data Presentation".
- Universality -- XML enables the representation of data in a manner that can be self-describing and thus universally used
- Persistence -- Through the materialization of data as an XML document this data can persist while still allowing programmatic access and manipulation.
- Platform and application independence
- Scalability

Additional XML Resources

Here are some additional resources for information about XML:

- *The Oracle XML Handbook*, Ben Chang, Mark Scardina, et.al., Oracle Press
- *Building Oracle XML Applications*, Steve Muench, O'Reilly
- *XML Bible*, Elliotte Rusty Harold, IDG Books Worldwide
- *XML Unleashed, Morrison et al.*, SAMS
- *Building XML Applications*, St.Laurent and Cerami, McGraw-Hill
- *Building Web Sites with XML*, Michael Floyd, Prentice Hall PTR
- *Building Corporate Portals with XML*, Finkelstein and Aiken, McGraw-Hill
- *XML in a Nutshell*, O'Reilly
- *Learning XML - (Guide to) Creating Self-Describing Data*, Ray, O'Reilly
- http://www.xml.com/pub/rg/46
- http://www.xml.org/xmlorg_resources/index.shtml
- http://www.xmlmag.com/
- http://www.webmethods.com/
- http://www.infoshark.com/default2.htm
- http://www.clarient.org/

- http://www.xmlwriter.com/
- http://webdevelopersjournal.com/articles/why_xml.html
- http://www.w3schools.com/xml/
- http://www.w3scripts.com/xml/default.asp
- http://www.xml101.com/examples/
- http://www.w3.org/TR/REC-xml
- http://msdn.microsoft.com/xml/default.asp
- http://www.w3.org/TR lists W3C technical reports
- http://www.w3.org/xml is the W3C XML activity overview page
- http://www.xml.com includes latest industry news about xml
- http://www.xml-cml.org has information about Chemical Markup Language (CML). CML documents can be viewed and edited on the Jumbo browser.
- http://www.loc.gov/ead/ Encoded Archival Description (EAD) information developed for the US Library of Congress.
- http://www.docuverse.com/xlf for information about Extensible Log Format (XLF) a project to convert log files into XML log files to simplify log file administration.
- http://www.w3.org/Math for information about MathML which provides a way of interchanging equations between applications.
- http://www.naa.org Newspaper Association of America (naa) 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.

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