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

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

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS
Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are “commercial computer software” or “commercial technical data” pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software–Restricted Rights (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

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

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.
Contents

Send Us Your Comments

Preface

Part 1 User’s Guide

1 XML Publisher Introduction

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1-1</td>
</tr>
<tr>
<td>Process Overview</td>
<td>1-3</td>
</tr>
<tr>
<td>Structure of this Manual</td>
<td>1-4</td>
</tr>
</tbody>
</table>

2 Creating an RTF Template

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>2-1</td>
</tr>
<tr>
<td>Supported Modes</td>
<td>2-2</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>2-2</td>
</tr>
<tr>
<td>Overview</td>
<td>2-2</td>
</tr>
<tr>
<td>Using the XML Publisher Template Builder</td>
<td>2-3</td>
</tr>
<tr>
<td>Associating the XML Data to the Template Layout</td>
<td>2-3</td>
</tr>
<tr>
<td>Designing the Template Layout</td>
<td>2-6</td>
</tr>
<tr>
<td>Adding Markup to the Template Layout</td>
<td>2-6</td>
</tr>
<tr>
<td>Creating Placeholders</td>
<td>2-6</td>
</tr>
<tr>
<td>Defining Groups</td>
<td>2-10</td>
</tr>
<tr>
<td>Defining Headers and Footers</td>
<td>2-13</td>
</tr>
<tr>
<td>Native Support</td>
<td>2-13</td>
</tr>
<tr>
<td>Images and Charts</td>
<td>2-14</td>
</tr>
<tr>
<td>Images</td>
<td>2-14</td>
</tr>
<tr>
<td>Chart Support</td>
<td>2-15</td>
</tr>
<tr>
<td>Drawing, Shape and Clip Art Support</td>
<td>2-26</td>
</tr>
<tr>
<td>Supported Native Formatting Features</td>
<td>2-37</td>
</tr>
<tr>
<td>General Features</td>
<td>2-37</td>
</tr>
<tr>
<td>Alignment</td>
<td>2-38</td>
</tr>
<tr>
<td>Tables</td>
<td>2-38</td>
</tr>
<tr>
<td>Date Fields</td>
<td>2-40</td>
</tr>
<tr>
<td>Multicolumn Page Support</td>
<td>2-41</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Background and Watermark Support</td>
<td>2-42</td>
</tr>
<tr>
<td>Template Features</td>
<td>2-44</td>
</tr>
<tr>
<td>Page Breaks</td>
<td>2-44</td>
</tr>
<tr>
<td>Initial Page Number</td>
<td>2-45</td>
</tr>
<tr>
<td>Last Page Only Content</td>
<td>2-46</td>
</tr>
<tr>
<td>Hyperlinks</td>
<td>2-48</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>2-51</td>
</tr>
<tr>
<td>Generating PDF Bookmarks</td>
<td>2-51</td>
</tr>
<tr>
<td>Check Boxes</td>
<td>2-51</td>
</tr>
<tr>
<td>Drop Down Lists</td>
<td>2-53</td>
</tr>
<tr>
<td>Conditional Formatting</td>
<td>2-55</td>
</tr>
<tr>
<td>If Statements</td>
<td>2-55</td>
</tr>
<tr>
<td>If Statements in Boilerplate Text</td>
<td>2-56</td>
</tr>
<tr>
<td>If-then-Else Statements</td>
<td>2-57</td>
</tr>
<tr>
<td>Choose Statements</td>
<td>2-57</td>
</tr>
<tr>
<td>Column Formatting</td>
<td>2-59</td>
</tr>
<tr>
<td>Row Formatting</td>
<td>2-61</td>
</tr>
<tr>
<td>Cell Highlighting</td>
<td>2-62</td>
</tr>
<tr>
<td>Page-Level Calculations</td>
<td>2-64</td>
</tr>
<tr>
<td>Displaying Page Totals</td>
<td>2-64</td>
</tr>
<tr>
<td>Brought Forward/Carried Forward Totals</td>
<td>2-67</td>
</tr>
<tr>
<td>Running Totals</td>
<td>2-70</td>
</tr>
<tr>
<td>Data Handling</td>
<td>2-71</td>
</tr>
<tr>
<td>Sorting</td>
<td>2-71</td>
</tr>
<tr>
<td>Checking for Nulls</td>
<td>2-71</td>
</tr>
<tr>
<td>Regrouping the XML Data</td>
<td>2-72</td>
</tr>
<tr>
<td>Variables, Parameters, and Properties</td>
<td>2-79</td>
</tr>
<tr>
<td>Using Variables</td>
<td>2-79</td>
</tr>
<tr>
<td>Defining Parameters</td>
<td>2-79</td>
</tr>
<tr>
<td>Setting Properties</td>
<td>2-81</td>
</tr>
<tr>
<td>Advanced Report Layouts</td>
<td>2-83</td>
</tr>
<tr>
<td>Batch Reports</td>
<td>2-83</td>
</tr>
<tr>
<td>Cross-Tab Support</td>
<td>2-85</td>
</tr>
<tr>
<td>Dynamic Data Columns</td>
<td>2-88</td>
</tr>
<tr>
<td>Number and Date Formatting</td>
<td>2-91</td>
</tr>
<tr>
<td>Calendar and Time Zone Support</td>
<td>2-102</td>
</tr>
<tr>
<td>Using External Fonts</td>
<td>2-103</td>
</tr>
<tr>
<td>Advanced Design Options</td>
<td>2-104</td>
</tr>
<tr>
<td>XPath Overview</td>
<td>2-104</td>
</tr>
<tr>
<td>Namespace Support</td>
<td>2-107</td>
</tr>
<tr>
<td>Using the Context Commands</td>
<td>2-107</td>
</tr>
<tr>
<td>Using XSL Elements</td>
<td>2-110</td>
</tr>
<tr>
<td>Using FO Elements</td>
<td>2-112</td>
</tr>
<tr>
<td>Best Practices</td>
<td>2-112</td>
</tr>
</tbody>
</table>
3 Creating a PDF Template

PDF Template Overview .................................................. 3-1
Supported Modes .......................................................... 3-2
Designing the Layout ...................................................... 3-2
Adding Markup to the Template Layout ......................... 3-4
  Creating a Placeholder ................................................ 3-5
  Defining Groups of Repeating Fields ......................... 3-8
Adding Page Numbers and Page Breaks ....................... 3-9
Performing Calculations .............................................. 3-13
Completed PDF Template ............................................. 3-14
Runtime Behavior ........................................................ 3-15
Creating a Template from a Downloaded PDF ............ 3-15

4 eText Templates

Introduction ........................................................................ 4-1
Structure of eText Templates ........................................... 4-2
Constructing the Data Tables ........................................... 4-5
  Command Rows .......................................................... 4-5
  Structure of the Data Rows ....................................... 4-9
Setup Command Tables .................................................. 4-12
Expressions, Control Structure, and Functions .............. 4-20
Identifiers, Operators, and Literals ............................... 4-22

5 Using the Template Manager

Introduction ........................................................................ 5-1
Creating the Data Definition .......................................... 5-2
  Viewing and Updating a Data Definition ................... 5-3
Creating the Template .................................................... 5-4
  Copying a Template .................................................... 5-6
Viewing and Updating a Template .................................... 5-6
  Updating the Template General Definitions ............... 5-7
  Previewing a Template ............................................... 5-7
  Editing the Template Layout .................................... 5-7
  Adding Localized Templates for Additional Languages .... 5-8
  Mapping PDF Template Fields .................................. 5-8
  Setting Runtime Properties for a Template .................. 5-9
Translatable Templates ................................................... 5-9

6 Generating Your Customized Report

Using the Concurrent Manager to Generate Your Custom Output .... 6-1
7 XML Publisher Extended Functions
   Extended SQL and XSL Functions ............................................. 7-1
   XSL Equivalents ........................................................................ 7-4
   Using FO Elements .................................................................. 7-5

Part 2 Implementation and Developer’s Guide

8 Administration
   XML Publisher Administration .................................................. 8-1
   Configuration .......................................................................... 8-2
   Uploading Font Files ................................................................ 8-10
   Creating Font Mappings ......................................................... 8-10
   Locales .................................................................................... 8-12
   Font Fallback Logic .................................................................. 8-12
   Font File Location ................................................................. 8-13
   Predefined Fonts ...................................................................... 8-13
   Defining Currency Formats .................................................... 8-16

9 Data Templates
   Introduction ............................................................................... 9-1
   The Data Template Definition ................................................. 9-2
   Constructing the Data Template ............................................. 9-5
   How to Call a Data Template ................................................ 9-20
   Distributed Queries .................................................................. 9-25
   Sample Data Templates .......................................................... 9-27

10 Calling XML Publisher APIs
    Introduction .............................................................................. 10-1
    XML Publisher Core APIs ..................................................... 10-2
    PDF Form Processing Engine ................................................. 10-3
    RTF Processor Engine ........................................................... 10-9
    FO Processor Engine .............................................................. 10-10
    PDF Document Merger .......................................................... 10-21
    PDF Book Binder Processor .................................................. 10-28
    Document Processor Engine .................................................. 10-31
    Bursting Engine ..................................................................... 10-43
    XML Publisher Properties ..................................................... 10-53
    Applications Layer APIs ...................................................... 10-56
    Datasource APIs ................................................................. 10-57
    Template APIs ....................................................................... 10-59

11 Delivery Manager
   Introduction .............................................................................. 11-1
Delivering Documents via e-Mail ........................................... 11-2
Delivering Your Document to a Printer ................................ 11-6
Delivering Your Documents via Fax ....................................... 11-9
Delivering Your Documents to WebDAV Servers .................... 11-10
Deliver Your Documents Using FTP ..................................... 11-13
Delivering Documents over Secure FTP ................................ 11-14
Delivering Documents over HTTP ....................................... 11-18
Delivering Documents via AS2 ........................................... 11-20
Direct and Buffering Modes ............................................... 11-24
Monitoring Delivery Status ............................................... 11-26
Global Properties .................................................................. 11-26
Delivering Multiple Requests with a Single Output Stream ....... 11-27
Adding a Custom Delivery Channel ..................................... 11-28
Configuration File Support .................................................. 11-34
Setting Up CUPS .................................................................. 11-36

12 Integrating the Document Viewer into an Application
  Overview ............................................................................. 12-1
  Parameters ........................................................................... 12-1
  Implementing the Document Viewer in an Application Page .... 12-3
  Document Viewer Common Region APIs ............................. 12-5

A XML Publisher Configuration File
  XML Publisher Configuration File ........................................ A-1
    Structure .......................................................................... A-2
    Properties ................................................................. A-3
    List of Available Properties ........................................ A-4
    Font Definitions ......................................................... A-4
    Font Fallback Mechanism ............................................ A-6

B Moving Templates and Data Definitions Between E-Business Suite Instances
  Overview ................................................................. B-1

C Supported XSL-FO Elements
  Supported XSL-FO Elements ........................................... C-1

Index
Send Us Your Comments

Oracle XML Publisher User’s Guide, Release 11i
Part No. B13817-05

Oracle welcomes your comments and suggestions on the quality and usefulness of this publication. 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 about this manual?

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

• Electronic mail: appsdoc_us@oracle.com
• FAX: 650-506-7200 Attn: Oracle Applications Technology Group Documentation Manager
• Postal service:
  Oracle Applications Technology Group Documentation Manager
  Oracle Corporation
  500 Oracle Parkway
  Redwood Shores, CA 94065
  USA

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

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

Intended Audience

Welcome to Release 11i of the Oracle XML Publisher User’s Guide.

This manual is intended to instruct users on how to use Oracle XML Publisher and common desktop tools to create customized reports.

If you are an E-Business Suite customer, this guide assumes you have a working knowledge of the following:

- The principles and customary practices of your business area.
- Standard request submission in Oracle Applications.
- The Oracle E-Business Suite user interfaces.

To learn more about standard request submission and the Oracle E-Business Suite graphical user interfaces, read the Oracle Applications User’s Guide.

If you have never used Oracle Applications, Oracle suggests you attend one or more of the Oracle Applications training classes available through Oracle University.

See Related Documents on page xii for more Oracle Applications product information.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility 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 more information, visit the Oracle Accessibility Program Web site at http://www.oracle.com/accessibility/.

Accessibility of Code Examples in Documentation

Screen readers 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, some screen readers may not always read a line of text that consists solely of a bracket or brace.

**Accessibility of Links to External Web Sites in Documentation**

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

**Structure**

1. XML Publisher Introduction
2. Creating an RTF Template
3. Creating a PDF Template
4. eText Templates
5. Using the Template Manager

This chapter pertains to Oracle E-Business Suite installations only.

6. Generating Your Customized Report

This chapter pertains to Oracle E-Business Suite installations only.

7. XML Publisher Extended Functions
8. Administration
9. Data Templates
10. Calling XML Publisher APIs
11. Delivery Manager
12. Integrating the Document Viewer into an Application

A. XML Publisher Configuration File
B. Moving Templates and Data Definitions Between E-Business Suite Instances
C. Supported XSL-FO Elements

**Related Documents**

**Online Documentation**

All Oracle Applications documentation is available online (HTML or PDF).

- **PDF Documentation** - See the Online Documentation CD for current PDF documentation for your product with each release. This Documentation CD is also available on OracleMetaLink and is updated frequently.

- **Online Help** - You can refer to Oracle Applications Help for current HTML online help for your product. Oracle provides patchable online help, which you can apply to your system for updated implementation and end user documentation. No system downtime is required to apply online help.

- **Release Content Document** - See the Release Content Document for descriptions of new features available by release. The Release Content Document is available on OracleMetaLink.

- **About document** - Refer to the About document for information about your release, including feature updates, installation information, and new documentation or documentation patches that you can download. The About document is available on OracleMetaLink.
Related Guides

Oracle Applications shares business and setup information with other Oracle Applications products. Therefore, you may want to refer to other guides when you set up and use Oracle Applications.

You can read the guides online by choosing Library from the expandable menu on your HTML help window, by reading from the Oracle Applications Document Library CD included in your media pack, or by using a Web browser with a URL that your system administrator provides.

If you require printed guides, you can purchase them from the Oracle Store at http://oraclestore.oracle.com.

Documents Related to this Product

Oracle Applications User's Guide

This guide explains how to enter data, query, run reports, and navigate using the graphical user interface (GUI). This guide also includes information on setting user profiles, as well as running and reviewing reports and concurrent processes.

Oracle Applications System Administrator's Guide

This guide provides planning and reference information for the Oracle Applications System Administrator. It contains information on how to define security and users, set report output definitions, and manage concurrent processing.

"About" Document

For information about implementation and user documentation, instructions for applying patches, new and changed setup steps, and descriptions of software updates, refer to the "About" document for your product. "About" documents are available on OracleMetaLink for most products starting with Release 11.5.8.

Do Not Use Database Tools to Modify Oracle Applications Data

Oracle STRONGLY RECOMMENDS that you never use SQL*Plus, Oracle Data Browser, database triggers, or any other tool to modify Oracle Applications data unless otherwise instructed.

Oracle provides powerful tools you can use to create, store, change, retrieve, and maintain information in an Oracle database. But if you use Oracle tools such as SQL*Plus to modify Oracle Applications data, you risk destroying the integrity of your data and you lose the ability to audit changes to your data.

Because Oracle Applications tables are interrelated, any change you make using an Oracle Applications form can update many tables at once. But when you modify Oracle Applications data using anything other than Oracle Applications, you may change a row in one table without making corresponding changes in related tables. If your tables get out of synchronization with each other, you risk retrieving erroneous information and you risk unpredictable results throughout Oracle Applications.

When you use Oracle Applications to modify your data, Oracle Applications automatically checks that your changes are valid. Oracle Applications also keeps track of who changes information. If you enter information into database tables using database tools, you may store invalid information. You also lose the ability to track who has changed your information because SQL*Plus and other database tools do not keep a record of changes.
This chapter covers the following topics:

- Introduction
- Process Overview
- Structure of this Manual

Introduction

Oracle XML Publisher is a template-based publishing solution delivered with the Oracle E-Business Suite. It provides a new approach to report design and publishing by integrating familiar desktop word processing tools with existing E-Business Suite data reporting. XML Publisher leverages standard, well-known technologies and tools, so you can rapidly develop and maintain custom report formats.

The flexibility of XML Publisher is a result of the separation of the presentation of the report from its data structure. The collection of the data is still handled by the E-Business Suite, but now you can design and control how the report outputs will be presented in separate template files. At runtime, XML Publisher merges your designed template files with the report data to create a variety of outputs to meet a variety of business needs, including:

- Customer-ready PDF documents, such as financial statements, marketing materials, contracts, invoices, and purchase orders utilizing colors, images, font styles, headers and footers, and many other formatting and design options.
- HTML output for optimum online viewing.
- Excel output to create a spreadsheet of your report data.
- "Filled-out" third-party provided PDF documents. You can download a PDF document, such as a government form, to use as a template for your report. At runtime, the data and template produce a "filled-out" form.
- Flat text files to exchange with business partners for EDI and EFT transmission.

The following graphic displays a few sample documents generated by XML Publisher:
User Interfaces

XML Publisher provides the Template Manager for E-Business Suite users to register and maintain report templates and their data sources. Once both have been registered, use the XML Publisher Concurrent Request to merge the template and its data source into the customized report.

**Note:** The Oracle Application Object Library (FND) patch 3435480 fully integrates XML Publisher with standard request submission both in Oracle Forms and HTML-based applications. If you have taken this upgrade, or are running Oracle E-Business Suite 11.5.10 Cumulative Update 1 (or later) you are not required to run the XML Publisher Concurrent Request.

Template Manager

The Template Manager is the repository for your templates and data sources. It is also the interface for you to associate your templates to data definitions and make them available to XML Publisher at runtime. From the Template Manager you can download, update, and preview your templates and translations.

XML Report Publisher Concurrent Request
The XML Report Publisher concurrent request produces the final output of your customized report. Before running this request, run your E-Business Suite report to obtain the XML data file. The XML Report Publisher request accepts as parameters the E-Business Suite report request ID and the desired template. The template must be associated to the report data definition in the Template Manager. The XML Report Publisher request merges the data and the template.

**XML Publisher Desktop Components**

XML Publisher provides components that you can install on your desktop to facilitate template development.

The Template Builder is an extension to Microsoft Word that simplifies the development of RTF templates. It automates many of the manual steps that would otherwise be required.

The Template Viewer is a Java application that facilitates the rapid development of templates by providing advanced preview capabilities for all template types.

Please see OracleMetaLink note 357308.1, "About Oracle XML Publisher Release 5.6.1” for the patch number for the Oracle XML Publisher Desktop components. See the patch readme and help files for user documentation and desktop system requirements.

**Process Overview**

Creating customized reports using XML Publisher can be divided into two phases: Design Time and Runtime.

**Design Time**

1. Register the E-Business Suite report as a Data Definition in the Template Manager.
   
   Create a Data Definition in the Template Manager for E-Business Suite reports that you wish to customize using XML Publisher. When you create the Data Definition, the Data Definition Code must match the E-Business Suite report shortname.

2. Design your template.
   
   Your template files can be either in Rich Text Format (RTF) or Portable Document Format (PDF).

   RTF is a specification used by many word processing applications, such as Microsoft Word. You design the template using your desktop word processing application and save the file as an RTF file type (.rtf extension). Insert basic markup tags to the document to prepare it for merging with the XML data. XML Publisher recognizes the formatting features that you apply and converts them to XSL-FO.

   Use Adobe Acrobat to apply markup tags to your custom-designed or downloaded PDF template.

3. Register your Template in the Template Manager.
   
   When you create the template in the Template Manager, you register and upload your RTF or PDF template files. The Template must be assigned to the Data Definition Code of the E-Business Suite report with which it will be merged.

4. Add desired translations of your RTF template.
XML Publisher’s translation utility allows you to extract the translatable strings from your template into an XLIFF file. Translate the strings in this file and reupload to the Template Manager to make the translation available at runtime.

**Runtime**

1. Set the concurrent program to generate XML.
2. Run the concurrent program using standard request submission to obtain the XML output.

   **Note:** The Application Object Library (FND) patch 3435480, Publishing Concurrent Requests with XML Publisher, fully integrates XML Publisher with the concurrent manager’s standard request submission. Simply run the request and select your template from the Submit Request interface and the concurrent manager will call XML Publisher to merge the template with the data all in a single step. You are not required to run the XML Publisher Concurrent request. This functionality is also available in the Oracle E-Business Suite 11.5.10 Cumulative Update 1.

3. Run the XML Publisher Concurrent Request.

   The XML Publisher Concurrent Request will prompt you to enter the Request ID from the previous step, and to select a template, template locale, and output type. Available templates are those associated to the report Data Definition in the Template Manager. XML Publisher merges your design template with the XML data to generate your customized output.

**Structure of this Manual**

This manual is divided into a User’s Guide and a Developer’s Guide. It contains the following information to enable you to get started and fully implement the capabilities of XML Publisher.

**User’s Guide**

This section includes instructions for designing templates, using the Template Manager, and generating your report output.

Creating an RTF Template - describes how to use your word processing application in conjunction with your report XML file to create a customized template for the report.

Creating a PDF Template - describes how to use Adobe Acrobat in conjunction with your report XML file to create a customized template in PDF.

eText Templates - describes how to create a table-based template to comply with EDI and EFT file specifications. These templates are processed by the eText Processing Engine to create flat text files for exchange with business partners.

Using the Template Manager - (pertains to E-Business Suite customers only) describes how to register your Oracle report as a data definition and upload your templates to the Template Manager.

Generating Your Customized Output - (pertains to E-Business Suite customers only) describes how to submit your report request using the concurrent manager to generate output in your customized template.
XML Publisher Extended Functions - lists SQL and XSL functions that XML Publisher has extended.

**Implementation and Developer’s Guide**

This section documents configuration information as well as programmatic interaction with XML Publisher for developers and standalone customers.

Administration - describes the Administration interface that allows you to set configuration properties, upload fonts, create font mappings, and create currency mappings.

Data Template - describes how to write a template to extract XML data using XML Publisher’s data engine.

Calling XML Publisher APIs - describes how to leverage XML Publisher’s processing engines via APIs.

Delivery Manager - describes how to use XML Publisher’s Delivery Manager APIs to deliver your documents via multiple channels, and how to create a custom channel.

Integrating the Document Viewer into an Application - describes how to implement XML Publisher’s document viewer, an Oracle Applications Framework component, in an application.

Moving Templates and Data Definitions Between E-Business Suite Instances - describes how to use the FNDLOAD and XDOLoader utilities to move your XML Publisher objects between test, development, and production instances.

XML Publisher Configuration File - describes how to set up a configuration file to set the Administration properties.

Supported XSL-FO Elements - lists the FO elements supported by the XML Publisher engines.
Creating an RTF Template

This chapter covers the following topics:

• Introduction
• Overview
• Designing the Template Layout
• Adding Markup to the Template Layout
• Defining Headers and Footers
• Images and Charts
• Drawing, Shape and Clip Art Support
• Supported Native Formatting Features
• Template Features
• Conditional Formatting
• Page-Level Calculations
• Data Handling
• Variables, Parameters, and Properties
• Advanced Report Layouts
• Number and Date Formatting
• Calendar and Time Zone Support
• Using External Fonts
• Advanced Design Options
• Best Practices

Introduction

Rich Text Format (RTF) is a specification used by common word processing applications, such as Microsoft Word. When you save a document, RTF is a file type option that you select.

XML Publisher’s RTF Template Parser converts documents saved as the RTF file type to XSL-FO. You can therefore create report designs using many of your standard word processing application’s design features and XML Publisher will recognize and maintain the design.
During design time, you add data fields and other markup to your template using XML Publisher’s simplified tags for XSL expressions. These tags associate the XML report data to your report layout. If you are familiar with XSL and prefer not to use the simplified tags, XML Publisher also supports the use of pure XSL elements in the template.

In addition to your word processing application’s formatting features, XML Publisher supports other advanced reporting features such as conditional formatting, dynamic data columns, running totals, and charts.

If you wish to include code directly in your template, you can include any XSL element, many FO elements, and a set of SQL expressions extended by XML Publisher.

**Supported Modes**

XML Publisher supports two methods for creating RTF templates:

- **Basic RTF Method**
  
  Use any word processing application that supports RTF version 1.6 writer (or later) to design a template using XML Publisher’s simplified syntax.

- **Form Field Method**
  
  Using Microsoft Word’s form field feature allows you to place the syntax in hidden form fields, rather than directly into the design of your template. XML Publisher supports Microsoft Word 2000 (or later) with Microsoft Windows version 2000 (or later).

  **Note:** If you use XSL or XSL:FO code rather than the simplified syntax, you must use the form field method.

This guide describes how to create RTF templates using both methods.

**Prerequisites**

Before you design your template, you must:

- Know the business rules that apply to the data from your source report.
- Generate a sample of your source report in XML.
- Be familiar with the formatting features of your word processing application.

**Overview**

Creating an RTF template file consists of two basic steps:

1. Design your template layout.
   
   Use the formatting features of your word processing application and save the file as RTF.

2. Mark up your template layout.
   
   Insert the XML Publisher simplified tags.

When you design your template layout, you must understand how to associate the XML input file to the layout. This chapter presents a sample template layout with its input XML file to illustrate how to make the proper associations to add the markup tags to the template.
Using the XML Publisher Template Builder

The Template Builder is an extension to Microsoft Word that simplifies the development of RTF templates. It automates many of the manual steps that are covered in this chapter. Use it in conjunction with this manual to increase your productivity.

The Template Builder is tightly integrated with Microsoft Word and allows you to perform the following functions:

- Insert data fields
- Insert data-driven tables
- Insert data-driven forms
- Insert data-driven charts
- Preview your template with sample XML data
- Browse and update the content of form fields
- Extract boilerplate text into an XLIFF translation file and test translations

Manual steps for performing these functions are covered in this chapter. Instructions and tutorials for using the Template Builder are available from the readme and help files delivered with the tool.

Note: If you are running XML Publisher through the E-Business Suite, please see OracleMetaLink note 357308.1, "About Oracle XML Publisher Release 5.6.1" for the patch number for the Oracle XML Publisher Desktop components. See the patch readme and help files for user documentation and desktop system requirements.

Associating the XML Data to the Template Layout

The following is a sample layout for a Payables Invoice Register:

**Sample Template Layout**

```
Supplier:

<table>
<thead>
<tr>
<th>Invoice Num</th>
<th>Invoice Date</th>
<th>GL Date</th>
<th>Curr</th>
<th>Entered Amt</th>
<th>Accounted Amt</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Total for Supplier:


```

Note the following:
The data fields that are defined on the template

For example: Supplier, Invoice Number, and Invoice Date

The elements of the template that will repeat when the report is run.

For example, all the fields on the template will repeat when the report is run. Each row of the invoice table will repeat for each invoice that is reported.

**XML Input File**

Following is the XML file that will be used as input to the Payables Invoice Register report template:

*Note:* To simplify the example, the XML output shown below has been modified from the actual output from the Payables report.

```xml
<?xml version="1.0" encoding="WINDOWS-1252" ?>
< VENDOR_REPORT 
  < LIST_G_VENDOR_NAME >
    < G_VENDOR_NAME >
      COMPANY A
    </ G_VENDOR_NAME >
    < LIST_G_INVOICE_NUM >
      < G_INVOICE_NUM >
        < SET_OF_BOOKS_ID >124</ SET_OF_BOOKS_ID >
        < GL_DATE >10-NOV-03</ GL_DATE >
        < INV_TYPE >Standard</ INV_TYPE >
        < INVOICE_NUM >031110</ INVOICE_NUM >
        < INVOICE_DATE >10-NOV-03</ INVOICE_DATE >
        < INVOICE_CURRENCY_CODE >EUR</ INVOICE_CURRENCY_CODE >
        < ENT_AMT >122</ ENT_AMT >
        < ACCTD_AMT >122</ ACCTD_AMT >
        < VAT_CODE >VAT22%</ VAT_CODE >
      </ G_INVOICE_NUM >
    </ LIST_G_INVOICE_NUM >
    < ENT_SUM_VENDOR >1000.00</ ENT_SUM_VENDOR >
    < ACCTD_SUM_VENDOR >1000.00</ ACCTD_SUM_VENDOR >
  </ G_VENDOR_NAME >
</ LIST_G_VENDOR_NAME >
< ACCTD_SUM_REP >108763.68</ ACCTD_SUM_REP >
< ENT_SUM_REP >122039</ ENT_SUM_REP >
</ VENDOR_REPORT >
```

XML files are composed of elements. Each tag set is an element. For example, `< INVOICE_DATE ></ INVOICE_DATE >` is the invoice date element. "INVOICE_DATE" is the tag name. The data between the tags is the value of the element. For example, the value of INVOICE_DATE is "10-NOV-03”.

The elements of the XML file have a hierarchical structure. Another way of saying this is that the elements have parent-child relationships. In the XML sample, some elements are contained within the tags of another element. The containing element is the parent and the included elements are its children.

Every XML file has only one root element that contains all the other elements. In this example, VENDOR_REPORT is the root element. The elements LIST_G_VENDOR_NAME, ACCTD_SUM_REP, and ENT_SUM_REP are contained between the VENDOR_REPORT tags and are children of VENDOR_REPORT. Each child element can have child elements of its own.
Identifying Placeholders and Groups

Your template content and layout must correspond to the content and hierarchy of the input XML file. Each data field in your template must map to an element in the XML file. Each group of repeating elements in your template must correspond to a parent-child relationship in the XML file.

To map the data fields you define placeholders. To designate the repeating elements, you define groups.

Note: XML Publisher supports regrouping of data if your report requires grouping that does not follow the hierarchy of your incoming XML data. For information on using this feature, see Regrouping the XML Data, page 2-72.

Placeholders

Each data field in your report template must correspond to an element in the XML file. When you mark up your template design, you define placeholders for the XML elements. The placeholder maps the template report field to the XML element. At runtime the placeholder is replaced by the value of the element of the same name in the XML data file.

For example, the “Supplier” field from the sample report layout corresponds to the XML element VENDOR_NAME. When you mark up your template, you create a placeholder for VENDOR_NAME in the position of the Supplier field. At runtime, this placeholder will be replaced by the value of the element from the XML file (the value in the sample file is COMPANY A).

Identifying the Groups of Repeating Elements

The sample report lists suppliers and their invoices. There are fields that repeat for each supplier. One of these fields is the supplier’s invoices. There are fields that repeat for each invoice. The report therefore consists of two groups of repeating fields:

• Fields that repeat for each supplier
• Fields that repeat for each invoice

The invoices group is nested inside the suppliers group. This can be represented as follows:

Suppliers
• Supplier Name
• Invoices
  • Invoice Num
  • Invoice Date
  • GL Date
  • Currency
  • Entered Amount
  • Accounted Amount
• Total Entered Amount
• Total Accounted Amount
Compare this structure to the hierarchy of the XML input file. The fields that belong to the Suppliers group shown above are children of the element G_VENDOR_NAME. The fields that belong to the Invoices group are children of the element G_INVOICE_NUM.

By defining a group, you are notifying XML Publisher that for each occurrence of an element (parent), you want the included fields (children) displayed. At runtime, XML Publisher will loop through the occurrences of the element and display the fields each time.

**Designing the Template Layout**

Use your word processing application’s formatting features to create the design.

For example:

- Select the size, font, and alignment of text
- Insert bullets and numbering
- Draw borders around paragraphs
- Include a watermark
- Include images (jpg, gif, or png)
- Use table autoformatting features
- Insert a header and footer

For additional information on inserting headers and footers, see Defining Headers and Footers, page 2-13.

For a detailed list of supported formatting features in Microsoft Word, see Supported Native Formatting Features, page 2-37. Additional formatting and reporting features are described at the end of this section.

**Adding Markup to the Template Layout**

XML Publisher converts the formatting that you apply in your word processing application to XSL-FO. You add markup to create the mapping between your layout and the XML file and to include features that cannot be represented directly in your format.

The most basic markup elements are placeholders, to define the XML data elements; and groups, to define the repeating elements.

XML Publisher provides tags to add markup to your template.

*Note:* For the XSL equivalents of the XML Publisher tags, see XSL Equivalent Syntax, page 7-4.

**Creating Placeholders**

The placeholder maps the template field to the XML element data field. At runtime the placeholder is replaced by the value of the element of the same name in the XML data file.

Enter placeholders in your document using the following syntax:

```xml
<?XML element tag name?>
```

*Note:* The placeholder must match the XML element tag name exactly. It is case sensitive.
There are two ways to insert placeholders in your document:

1. **Basic RTF Method**: Insert the placeholder syntax directly into your template document.

2. **Form Field Method**: (Requires Microsoft Word) Insert the placeholder syntax in Microsoft Word’s Text Form Field Options window. This method allows you to maintain the appearance of your template.

### Basic RTF Method

Enter the placeholder syntax in your document where you want the XML data value to appear.

Enter the element’s XML tag name using the syntax:

```xml
<?XML element tag name?>
```

In the example, the template field “Supplier” maps to the XML element `VENDOR_NAME`. In your document, enter:

```xml
<?VENDOR_NAME?>
```

The entry in the template is shown in the following figure:

```
Supplier: <?VENDOR_NAME?>
```

### Form Field Method

Use Microsoft Word’s **Text Form Field Options** window to insert the placeholder tags:

1. Enable the **Forms** toolbar in your Microsoft Word application.
2. Position your cursor in the place you want to create a placeholder.
3. Select the **Text Form Field** toolbar icon. This action inserts a form field area in your document.
4. Double-click the form field area to invoke the **Text Form Field Options** dialog box.
5. (Optional) Enter a description of the field in the **Default text** field. The entry in this field will populate the placeholder’s position on the template.
   
   For the example, enter “Supplier 1”.
6. Select the **Add Help Text** button.
7. In the help text entry field, enter the XML element’s tag name using the syntax:

   ```xml
   <?XML element tag name?>
   ```

   You can enter multiple element tag names in the text entry field.
In the example, the report field "Supplier" maps to the XML element VENDOR_NAME. In the **Form Field Help Text** field enter:

\<?VENDOR_NAME?>

The following figure shows the **Text Form Field Options** dialog box and the **Form Field Help Text** dialog box with the appropriate entries for the Supplier field.

**Tip:** For longer strings of XML Publisher syntax, use the Help Key (F1) tab instead of the Status Bar tab. The text entry field on the Help Key (F1) tab allows more characters.

8. Select **OK** to apply.

The **Default text** is displayed in the form field on your template.

The figure below shows the Supplier field from the template with the added form field markup.
Complete the Example

The following table shows the entries made to complete the example. The Template Field Name is the display name from the template. The Default Text Entry is the value entered in the Default Text field of the Text Form Field Options dialog box (form field method only). The Placeholder Entry is the XML element tag name entered either in the Form Field Help Text field (form field method) or directly on the template.

<table>
<thead>
<tr>
<th>Template Field Name</th>
<th>Default Text Entry (Form Field Method)</th>
<th>Placeholder Entry (XML Tag Name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invoice Num</td>
<td>1234566</td>
<td>&lt;?INVOICE_NUM?&gt;</td>
</tr>
<tr>
<td>Invoice Date</td>
<td>1-Jan-2004</td>
<td>&lt;?INVOICE_DATE?&gt;</td>
</tr>
<tr>
<td>GL Date</td>
<td>1-Jan-2004</td>
<td>&lt;?GL_DATE?&gt;</td>
</tr>
<tr>
<td>Curr</td>
<td>USD</td>
<td>&lt;?INVOICE_CURRENCY_CODE?&gt;</td>
</tr>
<tr>
<td>Entered Amt</td>
<td>1000.00</td>
<td>&lt;?ENT_AMT?&gt;</td>
</tr>
<tr>
<td>Accounted Amt</td>
<td>1000.00</td>
<td>&lt;?ACCTD_AMT?&gt;</td>
</tr>
<tr>
<td>(Total of Entered Amt column)</td>
<td>1000.00</td>
<td>&lt;?ENT_SUM_VENDOR?&gt;</td>
</tr>
<tr>
<td>(Total of Accounted Amt column)</td>
<td>1000.00</td>
<td>&lt;?ACCTD_SUM_VENDOR?&gt;</td>
</tr>
</tbody>
</table>

The following figure shows the Payables Invoice Register with the completed form field placeholder markup.

See the Payables Invoice Register with Completed Basic RTF Markup, page 2-11 for the completed basic RTF markup.
Defining Groups

By defining a group, you are notifying XML Publisher that for each occurrence of an element, you want the included fields displayed. At runtime, XML Publisher will loop through the occurrences of the element and display the fields each time.

In the example, for each occurrence of G_VENDOR_NAME in the XML file, we want the template to display its child elements VENDOR_NAME (Supplier Name), G_INVOICE_NUM (the Invoices group), Total Entered Amount, and Total Accounted Amount. And, for each occurrence of G_INVOICE_NUM (Invoices group), we want the template to display Invoice Number, Invoice Date, GL Date, Currency, Entered Amount, and Accounted Amount.

To designate a group of repeating fields, insert the grouping tags around the elements to repeat.

Insert the following tag before the first element:

<?for-each:XML group element tag name?>

Insert the following tag after the final element:

<?end for-each?>

Grouping scenarios

Note that the group element must be a parent of the repeating elements in the XML input file.

- If you insert the grouping tags around text or formatting elements, the text and formatting elements between the group tags will be repeated.
- If you insert the tags around a table, the table will be repeated.
- If you insert the tags around text in a table cell, the text in the table cell between the tags will be repeated.
- If you insert the tags around two different table cells, but in the same table row, the single row will be repeated.
If you insert the tags around two different table rows, the rows between the tags will be repeated (this does not include the row that contains the "end group" tag).

**Basic RTF Method**

Enter the tags in your document to define the beginning and end of the repeating element group.

To create the Suppliers group in the example, insert the tag

```xml
<?for-each:G_VENDOR_NAME?>
```

before the Supplier field that you previously created.

Insert `<?end for-each?>` in the document after the summary row.

The following figure shows the Payables Invoice Register with the basic RTF grouping and placeholder markup:

![Payables Invoice Register](image)

```xml
<<?for-each:G_VENDOR_NAME?>><<?end for-each?>>
Supplier: <?VENDOR_NAME?>

<table>
<thead>
<tr>
<th>Invoice Num</th>
<th>Invoice Date</th>
<th>GL Date</th>
<th>Curr</th>
<th>Entered Amt</th>
<th>Accounted Amt</th>
</tr>
</thead>
</table>

Total for Supplier: <?VENDOR_NAME?>

<?ENT_SUM_VENDOR?> <?ACCTD_SUM_VENDOR?>
```

Form Field Method

1. Insert a form field to designate the beginning of the group.

   In the help text field enter:

   ```xml
   <?for-each:group element tag name?>
   ```

   To create the Suppliers group in the example, insert a form field before the Suppliers field that you previously created. In the help text field enter:

   ```xml
   <?for-each:G_VENDOR_NAME?>
   ```

   For the example, enter the Default text "Group: Suppliers" to designate the beginning of the group on the template. The Default text is not required, but can make the template easier to read.

2. Insert a form field after the final placeholder element in the group. In the help text field enter `<?end for-each?>`.

   For the example, enter the Default text "End: Suppliers" after the summary row to designate the end of the group on the template.
The following figure shows the template after the markup to designate the Suppliers group was added.

**Group: Suppliers**

**Supplier: Supplier 1**

<table>
<thead>
<tr>
<th>Invoice Num</th>
<th>Invoice</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234566</td>
<td>1-Jan-</td>
</tr>
</tbody>
</table>

**End Suppliers**

**Complete the Example**

The second group in the example is the invoices group. The repeating elements in this group are displayed in the table. For each invoice, the table row should repeat. Create a group within the table to contain these elements.

**Note:** For each invoice, only the table row should repeat, not the entire table. Placing the grouping tags at the beginning and end of the table row will repeat only the row. If you place the tags around the table, then for each new invoice the entire table with headings will be repeated.

To mark up the example, insert the grouping tag `<?for-each:G_INVOICE_NUM?>` in the table cell before the Invoice Num placeholder. Enter the Default text "Group:Invoices" to designate the beginning of the group.

Insert the end tag inside the final table cell of the row after the Accounted Amt placeholder. Enter the Default text "End:Invoices" to designate the end of the group.

The following figure shows the completed example using the form field method:
Defining Headers and Footers

Native Support

XML Publisher supports the use of the native RTF header and footer feature. To create a header or footer, use the your word processing application's header and footer insertion tools. As an alternative, or if you have multiple headers and footers, you can use start:body and end body tags to distinguish the header and footer regions from the body of your report.

Inserting Placeholders in the Header and Footer

At the time of this writing, Microsoft Word does not support form fields in the header and footer. You must therefore insert the placeholder syntax directly into the template (basic RTF method), or use the start body/end body syntax described in the next section.

Multiple or Complex Headers and Footers

If your template requires multiple headers and footers, create them by using XML Publisher tags to define the body area of your report. You may also want to use this method if your header and footer contain complex objects that you wish to place in form fields. When you define the body area, the elements occurring before the beginning of the body area will compose the header. The elements occurring after the body area will compose the footer.

Use the following tags to enclose the body area of your report:

```xml
<?start:body?>
<?end body?>
```

Use the tags either directly in the template, or in form fields.

The Payables Invoice Register contains a simple header and footer and therefore does not require the start body/end body tags. However, if you wanted to add another header to the template, define the body area as follows:
1. Insert `<start:body>` before the Suppliers group tag: `<for-each:G_VENDOR_NAME>`

2. Insert `<end body>` after the Suppliers group closing tag: `<end for-each>`

The following figure shows the Payables Invoice Register with the start body/end body tags inserted:

```
Payables Invoice Register
```

```
<?start:body?>
Group: Suppliers  Sorted: Sales
Supplier: Supplier 1

<table>
<thead>
<tr>
<th>Invoice Num</th>
<th>Invoice Date</th>
<th>GL Date</th>
<th>Curr</th>
<th>Entered Amt</th>
<th>Accounted Amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupInvoices 123456</td>
<td>1-Jan-2004</td>
<td>1-Jan-2004</td>
<td>USD</td>
<td>1000.00</td>
<td>1000.00</td>
</tr>
</tbody>
</table>

Total for Supplier: Supplier 1: 1000.00 1000.00
```

`<?end body?>`

**Different First Page and Different Odd and Even Page Support**

If your report requires a different header and footer on the first page of your report; or, if your report requires different headers and footers for odd and even pages, you can define this behavior using Microsoft Word's Page Setup dialog.

1. Select Page Setup from the File menu.
2. In the Page Setup dialog, select the Layout tab.
3. In the Headers and footers region of the dialog, select the appropriate check box:
   - Different odd and even
   - Different first page
4. Insert your headers and footers into your template as desired.

At runtime your generated report will exhibit the defined header and footer behavior.

**Images and Charts**

**Images**

XML Publisher supports several methods for including images in your published document:

**Direct Insertion**

Insert the jpg, gif, or png image directly in your template.

**URL Reference**

URL Reference
1. Insert a dummy image in your template.

2. In the **Format Picture** dialog box select the **Web** tab. Enter the following syntax in the **Alternative text** region to reference the image URL:

   ```
   url:{'http://image location'}
   ```

   For example, enter: ```url:{'http://www.oracle.com/images/ora_log.gif'}``` 

**OA Media Directory Reference**

**Note:** This method only applies to Oracle E-Business Suite installations.

1. Insert a dummy image in your template.

2. In the **Format Picture** dialog box select the **Web** tab. Enter the following syntax in the **Alternative text** region to reference the OA_MEDIA directory:

   ```
   url:{'${OA_MEDIA}/image name'}
   ```

   For example, enter:

   ```
   url:{'${OA_MEDIA}/ORACLE_LOGO.gif'}
   ```

**Element Reference from XML File**

1. Insert a dummy image in your template.

2. In the **Format Picture** dialog box select the **Web** tab. Enter the following syntax in the **Alternative text** region to reference the image URL:

   ```
   url:{IMAGE_LOCATION}
   ```

   where `IMAGE_LOCATION` is an element from your XML file that holds the full URL to the image.

   You can also build a URL based on multiple elements at runtime. Just use the `concat` function to build the URL string. For example:

   ```
   url:{concat(SERVER,'/',IMAGE_DIR,'/',IMAGE_FILE)}
   ```

   This method can also be used with the OA_MEDIA reference as follows:

   ```
   url:{concat('${OA_MEDIA}','/',IMAGE_FILE)}
   ```

**Chart Support**

XML Publisher leverages the graph capabilities of Oracle Business Intelligence Beans (BI Beans) to enable you to define charts and graphs in your RTF templates that will be populated with data at runtime. XML Publisher supports all the graph types and component attributes available from the BI Beans graph DTD.

The BI Beans graph DTD is fully documented in the following technical note available from the Oracle Technology Network [http://www.oracle.com/technology/index.html] (OTN): "DTD for Customizing Graphs in Oracle Reports.”

The following summarizes the steps to add a chart to your template. These steps will be discussed in detail in the example that follows:

1. Insert a dummy image in your template to define the size and position of your chart.
2. Add the definition for the chart to the Alternative text box of the dummy image. The chart definition requires XSL commands.

3. At runtime XML Publisher calls the BI Beans applications to render the image that is then inserted into the final output document.

Adding a Sample Chart

Following is a piece of XML data showing total sales by company division.

```xml
<sales year=2004>
<division>
    <name>Groceries</name>
    <totalsales>3810</totalsales>
    <costofsales>2100</costofsales>
</division>
<division>
    <name>Toys</name>
    <totalsales>2432</totalsales>
    <costofsales>1200</costofsales>
</division>
<division>
    <name>Cars</name>
    <totalsales>6753</totalsales>
    <costofsales>4100</costofsales>
</division>
<division>
    <name>Hardware</name>
    <totalsales>2543</totalsales>
    <costofsales>1400</costofsales>
</division>
<division>
    <name>Electronics</name>
    <totalsales>5965</totalsales>
    <costofsales>3560</costofsales>
</division>
</sales>
```

This example will show how to insert a chart into your template to display it as a vertical bar chart as shown in the following figure:
Note the following attributes of this chart:

- The style is a vertical bar chart.
- The chart displays a background grid.
- The components are colored.
- Sales totals are shown as Y-axis labels.
- Divisions are shown as X-axis labels.
- The chart is titled.
- The chart displays a legend.

Each of these properties can be customized to suit individual report requirements.

**Inserting the Dummy Image**

The first step is to add a dummy image to the template in the position you want the chart to appear. The image size will define how big the chart image will be in the final document.

**Important:** You must insert the dummy image as a "Picture" and not any other kind of object.

The following figure shows an example of a dummy image:
The image can be embedded inside a for-each loop like any other form field if you want the chart to be repeated in the output based on the repeating data. In this example, the chart is defined within the sales year group so that a chart will be generated for each year of data present in the XML file.

Right-click the image to open the Format Picture palette and select the Web tab. Use the Alternative text entry box to enter the code to define the chart characteristics and data definition for the chart.

**Adding Code to the Alternative Text Box**

The following graphic shows an example of the XML Publisher code in the Format Picture Alternative text box:
The content of the Alternative text represents the chart that will be rendered in the final document. For this chart, the text is as follows:
The first element of your chart text must be the chart: element to inform the RTF parser that the following code describes a chart object.

Next is the opening <Graph> tag. Note that the whole of the code resides within the tags of the <Graph> element. This element has an attribute to define the chart type: graphType. If this attribute is not declared, the default chart is a vertical bar chart. BI Beans supports many different chart types. Several more types are presented in this section. For a complete listing, see the BI Beans graph DTD documentation.

The following code section defines the chart type and attributes:

```
<Graph graphType = "BAR_VERT_CLUST">
  <Title text="Company Sales 2004" visible="true" horizontalAlignme
nt="CENTER"/>
  <Y1Title text="Sales in Thousands" visible="true"/>
  <O1Title text="Division" visible="true"/>
  <LocalGridData colCount="{count(//division)}" rowCount="1">
    <RowLabels>
      <Label>Total Sales $1000s</Label>
    </RowLabels>
    <ColLabels>
      <xsl:for-each select="//division">
        <Label>
          <xsl:value-of select="name"/>
        </Label>
      </xsl:for-each>
    </ColLabels>
    <DataValues>
      <RowData>
        <xsl:for-each select="//division">
          <Cell>
            <xsl:value-of select="totalsales"/>
          </Cell>
        </xsl:for-each>
      </RowData>
    </DataValues>
  </LocalGridData>
</Graph>
```

All of these values can be declared or you can substitute values from the XML data at runtime. For example, you can retrieve the chart title from an XML tag by using the following syntax:

```
<Title text="{CHARTTITLE}" visible="true" horizontalAlignme
nt="CENTER"/>
```

where "CHARTTITLE" is the XML tag name that contains the chart title. Note that the tag name is enclosed in curly braces.

The next section defines the column and row labels:
The LocalGridData element has two attributes: colCount and rowCount. These define the number of columns and rows that will be shown at runtime. In this example, a count function calculates the number of columns to render:

colCount="(count(/division))"

The rowCount has been hard-coded to 1. This value defines the number of sets of data to be charted. In this case it is 1.

Next the code defines the row and column labels. These can be declared, or a value from the XML data can be substituted at runtime. The row label will be used in the chart legend (that is, “Total Sales $1000s”).

The column labels for this example are derived from the data: Groceries, Toys, Cars, and so on. This is done using a for-each loop:

This code loops through the <division> group and inserts the value of the <name> element into the <Label> tag. At runtime, this will generate the following XML:

The next section defines the actual data values to chart:

Creating an RTF Template 2-21
Similar to the labels section, the code loops through the data to build the XML that is passed to the BI Beans rendering engine. This will generate the following XML:

```xml
<DataValues>
  <RowData>
    <Cell>3810</Cell>
    <Cell>2432</Cell>
    <Cell>6753</Cell>
    <Cell>2543</Cell>
    <Cell>5965</Cell>
  </RowData>
</DataValues>
```

**Additional Chart Samples**

You can also display this data in a pie chart as shown in the following figure:

![Company Sales 2004 Pie Chart](image)

The following is the code added to the template to render this chart at runtime:
Horizontal Bar Chart Sample

The following example shows total sales and cost of sales charted in a horizontal bar format. This example also adds the data from the cost of sales element (<costofsales>) to the chart:

**Company Sales 2004**

The following code defines this chart in the template:
To accommodate the second set of data, the `rowCount` attribute for the `LocalGridData` element is set to 2. Also note the `DataValues` section defines two sets of data: one for Total Sales and one for Cost of Sales.

**Changing the Appearance of Your Chart**

There are many attributes available from the BI Beans graph DTD that you can manipulate to change the look and feel of your chart. For example, the previous chart can be changed to remove the grid, place a graduated background, and change the bar colors and fonts as shown in the following figure:
The code to support this is as follows:

```
chart:
<Graph graphType = "BAR_HORIZ_CLUST">
<SeriesItems>
  <Series id="0" color="#ffcc00"/>
  <Series id="1" color="#ff6600"/>
</SeriesItems>
<O1MajorTick visible="false"/>
<X1MajorTick visible="false"/>
<Y1MajorTick visible="false"/>
<Y2MajorTick visible="false"/>
<MarkerText visible="true" markerTextPlace="MTP_CENTER"/>
<PlotArea borderTransparent="true">
  <SFX fillType="FT_GRADIENT" gradientDirection="GD_LEFT" gradientNumPins="300">
    <GradientPinStyle pinIndex="1" position="1" gradientPinLeftColor="#999999" gradientPinRightColor="#cc6600"/>
  </SFX>
</PlotArea>
<Title text="Company Sales 2004" visible="true">
  <GraphFont name="Tahoma" bold="false"/>
</Title>
```

The colors for the bars are defined in the `SeriesItems` section. The colors are defined in hexadecimal format as follows:
The following code hides the chart grid:

```xml
<OLMajorTick visible="false"/>
<X1MajorTick visible="false"/>
<Y1MajorTick visible="false"/>
<Y2MajorTick visible="false"/>
```

The MarkerText tag places the data values on the chart bars:

```xml
<MarkerText visible="true" markerTextPlace="MTP_CENTER"/>
```

The PlotArea section defines the background. The SFX element establishes the gradient and the borderTransparent attribute hides the plot border:

```xml
<PlotArea borderTransparent="true">
    <SFX fillType="FT_GRADIENT" gradientDirection="GD_LEFT"
        gradientNumPins="300">
        <GradientPinStyle pinIndex="1" position="1"
            gradientPinLeftColor="#999999"
            gradientPinRightColor="#cc6600"/>
    </SFX>
</PlotArea>
```

The Title text tag has also been updated to specify a new font type and size:

```xml
<Title text="Company Sales 2004" visible="true">
    <GraphFont name="Tahoma" bold="false"/>
</Title>
```

## Drawing, Shape and Clip Art Support

XML Publisher supports Microsoft Word drawing, shape, and clip art features. You can add these objects to your template and they will be rendered in your final PDF output.

The following AutoShape categories are supported:

- **Lines** - straight, arrowed, connectors, curve, free form, and scribble
- **Connectors** - straight connectors only are supported. Curved connectors can be achieved by using a curved line and specifying the end styles to the line.
- **Basic Shapes** - all shapes are supported.
- **Block arrows** - all arrows are supported.
- **Flowchart** - all flowchart objects are supported.
- **Stars and Banners** - all objects are supported.
- **Callouts** - the "line" callouts are not supported.
- **Clip Art** - add images to your templates using the Microsoft Clip Art libraries

### Freehand Drawing

Use the freehand drawing tool in Microsoft Word to create drawings in your template to be rendered in the final PDF output.

### Hyperlinks

You can add hyperlinks to your shapes. See Hyperlinks, page 2-48.
Layering
You can layer shapes on top of each other and use the transparency setting in Microsoft Word to allow shapes on lower layers to show through. The following graphic shows an example of layered shapes:

3-D Effects
XML Publisher does not currently support the 3-D option for shapes.

Microsoft Equation
Use the equation editor to generate equations in your output. The following figure shows an example of an equation:

\[ \sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})^2} \]

Organization Chart
Use the organization chart functionality in your templates and the chart will be rendered in the output. The following image shows an example of an organization chart:

```
Vice President of Operations

Director    Architect    Administrative Assistant
```
WordArt

You can use Microsoft Word’s WordArt functionality in your templates. The following graphic shows a WordArt example:

![XML Publisher]

Note: Some Microsoft WordArt uses a bitmap operation that currently cannot be converted to SVG. To use the unsupported WordArt in your template, you can take a screenshot of the WordArt then save it as an image (gif, jpeg, or png) and replace the WordArt with the image.

Data Driven Shape Support

In addition to supporting the static shapes and features in your templates, XML Publisher supports the manipulation of shapes based on incoming data or parameters, as well. The following manipulations are supported:

- Replicate
- Move
- Change size
- Add text
- Skew
- Rotate

These manipulations not only apply to single shapes, but you can use the group feature in Microsoft Word to combine shapes together and manipulate them as a group.

Placement of Commands

Enter manipulation commands for a shape in the Web tab of the shape’s properties dialog as shown in the following example figure:
Replicate a Shape

You can replicate a shape based on incoming XML data in the same way you replicate data elements in a for-each loop. To do this, use a for-each@shape command in conjunction with a shape-offset declaration. For example, to replicate a shape down the page, use the following syntax:

```xml
<?for-each@shape:SHAPE_GROUP?>
  <?shape-offset-y:(position()-1)*100?>
<?end for-each?>
```

where

- for-each@shape opens the for-each loop for the shape context
- SHAPE_GROUP is the name of the repeating element from the XML file. For each occurrence of the element SHAPE_GROUP a new shape will be created.
- shape-offset-y: - is the command to offset the shape along the y-axis.
- (position()-1)*100 - sets the offset in pixels per occurrence. The XSL position command returns the record counter in the group (that is 1,2,3,4); one is subtracted from that number and the result is multiplied by 100. Therefore for the first occurrence the offset would be 0: (1-1) * 100. The offset for the second occurrence would be 100 pixels: (2-1) *100. And for each subsequent occurrence the offset would be another 100 pixels down the page.

Add Text to a Shape

You can add text to a shape dynamically either from the incoming XML data or from a parameter value. In the property dialog enter the following syntax:

```xml
<?shape-text:SHAPETEXT?>
```
where SHAPETEXT is the element name in the XML data. At runtime the text will be inserted into the shape.

Add Text Along a Path

You can add text along a line or curve from incoming XML data or a parameter. After drawing the line, in the property dialog enter:

```xml
<?shape-text-along-path:SHAPETEXT?>
```

where SHAPETEXT is the element from the XML data. At runtime the value of the element SHAPETEXT will be inserted above and along the line.

Moving a Shape

You can move a shape or transpose it along both the x and y-axes based on the XML data. For example to move a shape 200 pixels along the y-axis and 300 along the x-axis, enter the following commands in the property dialog of the shape:

```xml
<?shape-offset-x:300?>
<?shape-offset-y:200?>
```

Rotating a Shape

To rotate a shape about a specified axis based on the incoming data, use the following command:

```xml
<?shape-rotate:ANGLE;'POSITION'?>
```

where

ANGLE is the number of degrees to rotate the shape. If the angle is positive, the rotation is clockwise; if negative, the rotation is counterclockwise.

POSITION is the point about which to carry out the rotation, for example, 'left/top'. Valid values are combinations of left, right, or center with center, top, or bottom. The default is left/top. The following figure shows these valid values:

![Diagram showing valid rotation points](image)

To rotate this rectangle shape about the bottom right corner, enter the following syntax:

```xml
<?shape-rotate:60,'right/bottom'?>
```
You can also specify an x,y coordinate within the shape itself about which to rotate.

**Skewing a Shape**

You can skew a shape along its x or y axis using the following commands:

```
<?shape-skew-x:ANGLE;'POSITION'?>
<?shape-skew-y:ANGLE;'POSITION'?>
```

where

ANGLE is the number of degrees to skew the shape. If the angle is positive, the skew is to the right.

POSITION is the point about which to carry out the rotation, for example, ’left/top’. Valid values are combinations of left, right, or center with center, top, or bottom. See the figure under Rotating a Shape, page 2-30. The default is ’left/top’.

For example, to skew a shape by 30 degrees about the bottom right hand corner, enter the following:

```
<?shape-skew-x:number(.)*30;'right/bottom'?>
```

**Changing the Size of a Shape**

You can change the size of a shape using the appropriate commands either along a single axis or both axes. To change a shape’s size along both axes, use:

```
<?shape-size:RATIO?>
```

where RATIO is the numeric ratio to increase or decrease the size of the shape. Therefore a value of 2 would generate a shape twice the height and width of the original. A value of 0.5 would generate a shape half the size of the original.

To change a shape’s size along the x or y axis, use:

```
<?shape-size-x:RATIO?>
<?shape-size-y:RATIO?>
```

Changing only the x or y value has the effect of stretching or shrinking the shape along an axis. This can be data driven.

**Combining Commands**

You can also combine these commands to carry out multiple transformations on a shape at one time. For example, you can replicate a shape and for each replication, rotate it by some angle and change the size at the same time.

The following example shows how to replicate a shape, move it 50 pixels down the page, rotate it by five degrees about the center, stretch it along the x-axis and add the number of the shape as text:

```
<for-each@shape:SHAPE_GROUP?>
  <?shape-text:position()?><br />
  <?shape-offset-y:position()*50?><br />
  <?shape-rotate:5;'center/center'?><br />
  <?shape-size-x:position()+1?><br />
<end for-each?>
```

This would generate the output shown in the following figure:
CD Ratings Example

This example demonstrates how to set up a template that will generate a star-rating based on data from an incoming XML file.

Assume the following incoming XML data:
Notice there is a USER_RATING element for each CD. Using this data element and the shape manipulation commands, we can create a visual representation of the ratings so that the reader can compare them at a glance.

A template to achieve this is shown in the following figure:

<table>
<thead>
<tr>
<th>Title</th>
<th>Artist</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empire Burlesque</td>
<td>Bob Dylan</td>
<td>4</td>
</tr>
<tr>
<td>Hide Your Heart</td>
<td>Bonnie Tyler</td>
<td>3</td>
</tr>
<tr>
<td>Still got the blues</td>
<td>Gary More</td>
<td>5</td>
</tr>
<tr>
<td>This is US</td>
<td>Gary Lee</td>
<td>2</td>
</tr>
</tbody>
</table>

The values for the fields are shown in the following table:
Field | Form Field Entry
--- | ---
F | `<?for-each:CD?>`
TITLE | `<?TITLE?>`
ARTIST | `<?ARTIST?>`
E | `<?end for-each?>`
(star shape) | Web Tab Entry:
 | `<?for-each@shape:xdoxslt:foreach_number($_XDOCTX, 0, USER_RATING, 1)?>`
 | `<?shape-offset-x: (position()-1)*25?>`
 | `<?end for-each?>`

The form fields hold the simple element values. The only difference with this template is the value for the star shape. The replication command is placed in the Web tab of the Format AutoShape dialog.

In the for-each@shape command we are using a command to create a "for...next loop" construct. We specify 1 as the starting number; the value of USER_RATING as the final number; and 1 as the step value. As the template loops through the CDs, we create an inner loop to repeat a star shape for every USER_RATING value (that is, a value of 4 will generate 4 stars). The output from this template and the XML sample is shown in the following graphic:

<table>
<thead>
<tr>
<th>Title</th>
<th>Artist</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empire Burlesque</td>
<td>Bob Dylan</td>
<td>⭐⭐⭐⭐⭐</td>
</tr>
<tr>
<td>Hide Your Heart</td>
<td>Bonnie Tyler</td>
<td>⭐⭐⭐⭐</td>
</tr>
<tr>
<td>Still got the blues</td>
<td>Gary More</td>
<td>⭐⭐⭐⭐⭐</td>
</tr>
<tr>
<td>This is US</td>
<td>Gary Lee</td>
<td>⭐⭐</td>
</tr>
</tbody>
</table>

**Grouped Shape Example**

This example shows how to combine shapes into a group and have them react to the incoming data both individually and as a group. Assume the following XML data:
<SALES>
  <SALE>
    <REGION>Americas</REGION>
    <SOFTWARE>1200</SOFTWARE>
    <HARDWARE>850</HARDWARE>
    <SERVICES>2000</SERVICES>
  </SALE>
  <SALE>
    <REGION>EMEA</REGION>
    <SOFTWARE>1000</SOFTWARE>
    <HARDWARE>800</HARDWARE>
    <SERVICES>1100</SERVICES>
  </SALE>
  <SALE>
    <REGION>APAC</REGION>
    <SOFTWARE>900</SOFTWARE>
    <HARDWARE>1200</HARDWARE>
    <SERVICES>1500</SERVICES>
  </SALE>
</SALES>

You can create a visual representation of this data so that users can very quickly understand the sales data across all regions. Do this by first creating the composite shape in Microsoft Word that you wish to manipulate. The following figure shows a composite shape made up of four components:

The shape consists of three cylinders: red, yellow, and blue. These will represent the data elements software, hardware, and services. The combined object also contains a rectangle that is enabled to receive text from the incoming data.

The following commands are entered into the Web tab:

**Red cylinder:** `<?shape-size-y:SOFTWARE div 1000;'left/bottom'?>`

**Yellow cylinder:** `<?shape-size-y:HARDWARE div 1000;'left/bottom'?>`

**Blue cylinder:** `<?shape-size-y:SERVICES div 1000;'left/bottom'?>`

The shape-size command is used to stretch or shrink the cylinder based on the values of the elements SOFTWARE, HARDWARE, and SERVICES. The value is divided by 1000 to set the stretch or shrink factor. For example, if the value is 2000, divide that by 1000 to get a factor of 2. The shape will generate as twice its current height.

The text-enabled rectangle contains the following command in its Web tab:

`<?shape-text:REGION?>`

At runtime the value of the REGION element will appear in the rectangle.
All of these shapes were then grouped together and in the Web tab for the grouped object, the following syntax is added:

```xml
<?for-each@shape:SALE?>
<?shape-offset-x: (position()-1)*110?>
<?end for-each?>
```

In this set of commands, the `for-each@shape` loops over the SALE group. The `shape-offset` command moves the next shape in the loop to the right by a specific number of pixels. The expression `(position()-1)` sets the position of the object. The `position()` function returns a record counter while in the loop, so for the first shape, the offset would be 1-1*100, or 0, which would place the first rendering of the object in the position defined in the template. Subsequent occurrences would be rendered at a 100 pixel offset along the x-axis (to the right).

At runtime three sets of shapes will be rendered across the page as shown in the following figure:

![Shapes rendered across the page](image)

To make an even more visually representative report, these shapes can be superimposed onto a world map. Just use the "Order" dialog in Microsoft Word to layer the map behind the grouped shapes.

**Microsoft Word 2000 Users:** After you add the background map and overlay the shape group, use the Grouping dialog to make the entire composition one group.

**Microsoft Word 2002/3 Users:** These versions of Word have an option under Tools > Options, General tab to "Automatically generate drawing canvas when inserting autoshapes". Using this option removes the need to do the final grouping of the map and shapes. We can now generate a visually appealing output for our report as seen in the following figure:
Supported Native Formatting Features

In addition to the features already listed, XML Publisher supports the following features of Microsoft Word.

General Features

- Large blocks of text
- Page breaks

To insert a page break, insert a Ctrl-Enter keystroke just before the closing tag of a group. For example if you want the template to start a new page for every Supplier in the Payables Invoice Register:

1. Place the cursor just before the Supplier group’s closing <?end for-each?> tag.
2. Press Ctrl-Enter to insert a page break.

At runtime each Supplier will start on a new page.

Using this Microsoft Word native feature will cause a single blank page to print at the end of your report output. To avoid this single blank page, use XML Publisher’s page break alias. See Special Features: Page Breaks, page 2-44.

- Page numbering

Insert page numbers into your final report by using the page numbering methods of your word processing application. For example, if you are using Microsoft Word:

1. From the Insert menu, select Page Numbers...
2. Select the Position, Alignment, and Format as desired.
At runtime the page numbers will be displayed as selected.

Alignment

Use your word processor’s alignment features to align text, graphics, objects, and tables.

**Note:** Bidirectional languages are handled automatically using your word processing application’s left/right alignment controls.

Tables

Supported table features include:

- Nested Tables
- Cell Alignment
  You can align any object in your template using your word processing application’s alignment tools. This alignment will be reflected in the final report output.
- Row spanning and column spanning
  You can span both columns and rows in your template as follows:
  1. Select the cells you wish to merge.
  2. From the **Table** menu, select **Merge Cells**.
  3. Align the data within the merged cell as you would normally.
  At runtime the cells will appear merged.
- Table Autoformatting
  XML Publisher recognizes the table autoformats available in Microsoft Word.
  1. Select the table you wish to format.
  2. From the **Table** menu, select **Autoformat**.
  3. Select the desired table format.
  At runtime, the table will be formatted using your selection.
- Cell patterns and colors
  You can highlight cells or rows of a table with a pattern or color.
  1. Select the cell(s) or table.
  2. From the **Table** menu, select **Table Properties**.
  3. From the **Table** tab, select the **Borders and Shading...** button.
  4. Add borders and shading as desired.
- Repeating table headers
  If your data is displayed in a table, and you expect the table to extend across multiple pages, you can define the header rows that you want to repeat at the start of each page.
  1. Select the row(s) you wish to repeat on each page.
  2. From the **Table** menu, select **Heading Rows Repeat**.
• Prevent rows from breaking across pages.

If you want to ensure that data within a row of a table is kept together on a page, you can set this as an option using Microsoft Word’s **Table Properties**.

1. Select the row(s) that you want to ensure do not break across a page.
2. From the **Table** menu, select **Table Properties**.
3. From the **Row** tab, deselect the check box "Allow row to break across pages".

• **Fixed-width columns**

To set the widths of your table columns:

1. Select a column and then select **Table > Table Properties**.
2. In the **Table Properties** dialog, select the **Column** tab.
3. Enable the **Preferred width** checkbox and then enter the width as a **Percent** or in **Inches**.
4. Select the **Next Column** button to set the width of the next column.

Note that the total width of the columns must add up to the total width of the table.

The following figure shows the **Table Properties** dialog:

![Table Properties Dialog](image)

• **Text truncation**

By default, if the text within a table cell will not fit within the cell, the text will be wrapped. To truncate the text instead, use the table properties dialog.
1. Place your cursor in the cell in which you want the text truncated.
2. Right-click your mouse and select Table Properties... from the menu, or navigate to Table > Table Properties...
3. From the Table Properties dialog, select the Cell tab, then select Options...
4. Deselect the Wrap Text check box.

The following figure shows the Cell Options dialog.

![Cell Options Dialog]

An example of truncation is shown in the following graphic:

<table>
<thead>
<tr>
<th>Wrap Text checked</th>
<th>Wrap Text unchecked</th>
</tr>
</thead>
<tbody>
<tr>
<td>The quick brown fox jumped over the lazy river.</td>
<td>The quick brown fox</td>
</tr>
</tbody>
</table>

**Date Fields**

Insert dates using the date feature of your word processing application. Note that this date will correspond to the publishing date, not the request run date.
**Multicolumn Page Support**

XML Publisher supports Microsoft Word’s Columns function to enable you to publish your output in multiple columns on a page.

Select **Format > Columns** to display the **Columns** dialog box to define the number of columns for your template. The following graphic shows the Columns dialog:

![Columns Dialog Box](image)

**Multicolumn Page Example: Labels**

To generate address labels in a two-column format:

1. Divide your page into two columns using the Columns command.
2. Define the repeatable group in the first column. Note that you define the repeatable group only in the first column, as shown in the following figure:

```
<table>
<thead>
<tr>
<th>Name</th>
<th>CUSTOMER_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>CUSTOMER_NUMBER</td>
</tr>
<tr>
<td>City</td>
<td>CITY</td>
</tr>
<tr>
<td>State</td>
<td>STATE</td>
</tr>
<tr>
<td>Zip Code</td>
<td>ZIP_CODE</td>
</tr>
</tbody>
</table>
```

**Tip:** To prevent the address block from breaking across pages or columns, embed the label block inside a single-celled table. Then specify in the Table Properties that the row should not break across pages. See Prevent rows from breaking across pages, page 2-39.
This template will produce the following multicolumn output:

<table>
<thead>
<tr>
<th>Name</th>
<th>Nuts and Bolts Ltd</th>
<th>Name</th>
<th>Big Co</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>1220</td>
<td>Number</td>
<td>1221</td>
</tr>
<tr>
<td>City</td>
<td>Espoo</td>
<td>City</td>
<td>Helsinki</td>
</tr>
<tr>
<td>State</td>
<td>FI</td>
<td>State</td>
<td>FI</td>
</tr>
<tr>
<td>Zip Code</td>
<td>Il...</td>
<td>Zip Code</td>
<td>Il...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>My Company</th>
<th>Name</th>
<th>Small Co</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>1220</td>
<td>Number</td>
<td>1221</td>
</tr>
<tr>
<td>City</td>
<td>Espoo</td>
<td>City</td>
<td>Helsinki</td>
</tr>
<tr>
<td>State</td>
<td>FI</td>
<td>State</td>
<td>FI</td>
</tr>
<tr>
<td>Zip Code</td>
<td>Il...</td>
<td>Zip Code</td>
<td>Il...</td>
</tr>
</tbody>
</table>
From this dialog select one of the following supported options:

- Gradient - this can be either one or two colors
- Texture - choose one of the textures provided, or load your own
- Pattern - select a pattern and background/foreground colors
- Picture - load a picture to use as a background image

**Add a Text or Image Watermark Using Microsoft Word 2002 or later**

These versions of Microsoft Word allow you to add either a text or image watermark. Use the Format > Background > Printed Watermark dialog to select either:

- Picture Watermark - load an image and define how it should be scaled on the document
- Text Watermark - use the predefined text options or enter your own, then specify the font, size and how the text should be rendered.

The following figure shows the Printed Watermark dialog completed to display a text watermark:
Template Features

Page Breaks

To create a page break after the occurrence of a specific element use the "split-by-page-break" alias. This will cause the report output to insert a hard page break between every instance of a specific element.

To insert a page break between each occurrence of a group, insert the "split-by-page-break" form field within the group immediately before the <?end for-each?> tag that closes the group. In the Help Text of this form field enter the syntax:

<?split-by-page-break:?>

Example
For the following XML, assume you want to create a page break for each new supplier:

```
<supplier name="2-44">
  <name>Oracle XML Publisher User's Guide</name>
</supplier>
```
In the template sample shown in the following figure, the field called PageBreak contains the split-by-page-break syntax:

<table>
<thead>
<tr>
<th>Invoice Number</th>
<th>Invoice Date</th>
<th>Amount</th>
<th>Running Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>FE10001-1</td>
<td>1-Jan-2005</td>
<td>100.00</td>
<td>100.00</td>
</tr>
<tr>
<td>PageBreak</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Place the PageBreak field with the `<?split-by-page-break:?>` syntax immediately before the `<?end for-each?>` field. The PageBreak field sits inside the end of the SUPPLIER loop. This will ensure a page break is inserted before the occurrence of each new supplier. This method avoids the ejection of an extra page at the end of the group when using the native Microsoft Word page break after the group.

**Initial Page Number**

Some reports require that the initial page number be set at a specified number. For example, monthly reports may be required to continue numbering from month to month. XML Publisher allows you to set the page number in the template to support this requirement.

Use the following syntax in your template to set the initial page number:

```xml
<?initial-page-number:pagenumber?>
```

where `pagenumber` is the XML element or parameter that holds the numeric value.

**Example 1 - Set page number from XML data element**

If your XML data contains an element to carry the initial page number, for example:
Enter the following in your template:

```xml
<?initial-page-number:PAGESTART?>
```

Your initial page number will be the value of the PAGESTART element, which in this case is 200.

**Example 2 - Set page number by passing a parameter value**

If you define a parameter called PAGESTART, you can pass the initial value by calling the parameter.

Enter the following in your template:

```xml
<?initial-page-number:$PAGESTART?>
```

**Note:** You must first declare the parameter in your template. See Defining Parameters in Your Template, page 2-79.

### Last Page Only Content

XML Publisher supports the Microsoft Word functionality to specify a different page layout for the first page, odd pages, and even pages. To implement these options, simply select **Page Setup** from the **File** menu, then select the **Layout** tab. XML Publisher will recognize the settings you make in this dialog.

However, Microsoft Word does not provide settings for a different last page only. This is useful for documents such as checks, invoices, or purchase orders on which you may want the content such as the check or the summary in a specific place only on the last page.

XML Publisher provides this ability. To utilize this feature, you must:

1. Create a section break in your template to ensure the content of the final page is separated from the rest of the report.
2. Insert the following syntax on the final page:

   ```xml
   <?start@last-page:body?>
   <end body?>
   ```

Any content on the page that occurs above or below these two tags will appear only on the last page of the report. Also, note that because this command explicitly specifies the content of the final page, any desired headers or footers previously defined for the report must be reinserted on the last page.

**Example**

This example uses the last page only feature for a report that generates an invoice listing with a summary to appear at the bottom of the last page.

Assume the following XML:
<xml version="1.0" encoding="WINDOWS-1252">
<INVOICELIST>
  <VENDOR>
    <VENDOR_NAME>Nuts and Bolts Limited</VENDOR_NAME>
    <ADDRESS>1 El Camino Real, Redwood City, CA 94065</ADDRESS>
    <INVOICE>
      <INV_TYPE>Standard</INV_TYPE>
      <INVOICE_NUM>981110</INVOICE_NUM>
      <INVOICE_DATE>10-NOV-04</INVOICE_DATE>
      <INVOICE_CURRENCY_CODE>EUR</INVOICE_CURRENCY_CODE>
      <ENT_AMT>122</ENT_AMT>
      <ACCTD_AMT>122</ACCTD_AMT>
      <VAT_CODE>VAT22%</VAT_CODE>
    </INVOICE>
  </VENDOR>
  <VENDOR>
    ... ...
    <INVOICE>
      ... ...
    </INVOICE>
  </VENDOR>
  <SUMMARY>
    <SUM_ENT_AMT>61435</SUM_ENT_AMT>
    <SUM_ACCTD_AMT>58264.68</SUM_ACCTD_AMT>
    <TAX_CODE>EU22%</TAX_CODE>
  </SUMMARY>
</INVOICELIST>

The report should show each VENDOR and their INVOICE data with a SUMMARY section that appears only on the last page, placed at the bottom of the page. The template for this is shown in the following figure:

Template Page One

<table>
<thead>
<tr>
<th>Invoice Type</th>
<th>Invoice Num</th>
<th>Invoice Date</th>
<th>Invoice Currency</th>
<th>Entered Amount</th>
<th>Accounted Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>F Invoice</td>
<td>120000</td>
<td>01-Jan-2006</td>
<td>USD</td>
<td>100</td>
<td>100 E</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<<insert section break>>
Insert a Microsoft Word section break (type: next page) on the first page of the template. For the final page, insert new line characters to position the summary table at the bottom of the page. The summary table is shown in the following figure:

**Last Page Only Layout**

![Last Page Placeholder]

**Tax Summary**

<table>
<thead>
<tr>
<th>Tax Code</th>
<th>Entered Amount</th>
<th>Accounted Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAT 18.5</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

In this example:

- The F and E components contain the for-each grouping statements.
- The grayed report fields are placeholders for the XML elements.
- The "Last Page Placeholder" field contains the syntax:

  ```
  <?start@last-page:body?>
  <?end body?>
  ```

  to declare the last page layout. Any content above or below this statement will appear on the last page only. The content above the statement is regarded as the header and the content below the statement is regarded as the footer.

If your reports contain headers and footers that you want to carry over onto the last page, you must reinsert them on the last page. For more information about headers and footers see Defining Headers and Footers, page 2-13.

You must insert a section break (type: next page) into the document to specify the last page layout. This example is available in the samples folder of the Oracle XML Publisher Template Builder for Word installation. (See "About Oracle XML Publisher 5.6.1," OracleMetaLink note 357308.1 for information regarding the XML Publisher Desktop components.

It is important to note that if the report is only one page in length, the first page layout will be used. If your report requires that a single page report should default to the last page layout (such as in a check printing implementation) then you can use the following alternate syntax for the "Last Page Placeholder" on the last page:

```
<?start@last-page-first:body?>
<?end body?>
```

Substituting this syntax will result in the last page layout for reports that are only one page long.

**Hyperlinks**

XML Publisher supports several different types of hyperlinks. The hyperlinks can be fixed or dynamic and can link to either internal or external destinations. Hyperlinks can also be added to shapes.
To insert static hyperlinks to either text or a shape, use your word processing application’s insert hyperlink feature:

1. Select the text or shape.
2. Use the right-mouse menu to select Hyperlink; or, select Hyperlink from the Insert menu.
3. Enter the URL using any of the methods provided on the Insert Hyperlink dialog box.

The following screenshot shows the insertion of a static hyperlink using Microsoft Word’s Insert Hyperlink dialog box.

![Insert Hyperlink dialog box](image)

If your input XML data includes an element that contains a hyperlink or part of one, you can create dynamic hyperlinks at runtime. In the Type the file or Web page name field of the Insert Hyperlink dialog box, enter the following syntax:

```
{URL_LINK}
```

where URL_LINK is the incoming data element name.

If you have a fixed URL that you want to add elements from your XML data file to construct the URL, enter the following syntax:

```
http://www.oracle.com?product={PRODUCT_NAME}
```

where PRODUCT_NAME is the incoming data element name.

In both these cases, at runtime the dynamic URL will be constructed.

The following figure shows the insertion of a dynamic hyperlink using Microsoft Word’s Insert Hyperlink dialog box. The data element SUPPLIER_URL from the incoming XML file will contain the hyperlink that will be inserted into the report at runtime.
You can also pass parameters at runtime to construct a dynamic URL.
Enter the parameter and element names surrounded by braces to build up the URL as follows:

```{SERVER_URL}{REPORT}/cstid={CUSTOMER_ID}
```
where `SERVER_URL` and `REPORT` are parameters passed to the template at runtime (note the `$` sign) and `CUSTOMER_ID` is an XML data element. This link may render as:


**Inserting Internal Links**

Insert internal links into your template using Microsoft Word’s Bookmark feature.

1. Position your cursor in the desired destination in your document.
2. Select **Insert > Bookmark**...
3. In the **Bookmark** dialog, enter a name for this bookmark, and select **Add**.
4. Select the text or shape in your document that you want to link back to the Bookmark target.
5. Use the right-mouse menu to select **Hyperlink**; or select **Hyperlink** from the **Insert** menu.
6. On the **Insert Hyperlink** dialog, select **Bookmark**.
7. Choose the bookmark you created from the list.

At runtime, the link will be maintained in your generated report.
Table of Contents

XML Publisher supports the table of contents generation feature of the RTF specification. Follow your word processing application’s procedures for inserting a table of contents.

XML Publisher also provides the ability to create dynamic section headings in your document from the XML data. You can then incorporate these into a table of contents.

To create dynamic headings:

1. Enter a placeholder for the heading in the body of the document, and format it as a "Heading", using your word processing application’s style feature. You cannot use form fields for this functionality.

   For example, you want your report to display a heading for each company reported. The XML data element tag name is <COMPANY_NAME>. In your template, enter <?COMPANY_NAME?> where you want the heading to appear. Now format the text as a Heading.

2. Create a table of contents using your word processing application’s table of contents feature.

   At runtime the TOC placeholders and heading text will be substituted.

Generating PDF Bookmarks

If you have defined a table of contents in your RTF template, you can use your table of contents definition to generate links in the Bookmarks tab in the navigation pane of your output PDF. The bookmarks can be either static or dynamically generated.

For information on creating the table of contents, see Table of Contents, page 2-51.

- To create links for a static table of contents:

   Enter the syntax:
   
   <?copy-to-bookmark:?>
   
   directly above your table of contents and
   
   <?end copy-to-bookmark:?>
   
   directly below the table of contents.

- To create links for a dynamic table of contents:

   If you have a dynamic table of contents in your, enter the syntax:

   <?convert-to-bookmark:?>
   
   directly above the table of contents and
   
   <?end convert-to-bookmark:?>
   
   directly below the table of contents.

Check Boxes

You can include a check box in your template that you can define to display as checked or unchecked based on a value from the incoming data.

To define a check box in your template:
1. Position the cursor in your template where you want the check box to display, and select the Check Box Form Field from the Forms tool bar (shown in the following figure).

![Check Box Form Field](image)

2. Right-click the field to open the Check Box Form Field Options dialog.

3. Specify the Default value as either Checked or Not Checked.

4. In the Form Field Help Text dialog, enter the criteria for how the box should behave. This must be a boolean expression (that is, one that returns a true or false result).

   For example, suppose your XML data contains an element called `<population>`. You want the check box to appear checked if the value of `<population>` is greater than 10,000. Enter the following in the help text field:

   `<?population>10000?>`

   This is displayed in the following figure:
Note that you do not have to construct an “if” statement. The expression is treated as an “if” statement.

See the next section for a sample template using a check box.

**Drop Down Lists**

XML Publisher allows you to use the drop-down form field to create a cross-reference in your template from your XML data to some other value that you define in the drop-down form field.

For example, suppose you have the following XML:

```xml
<countries>
  <country>
    <name>Chad</name>
    <population>7360000</population>
    <continentIndex>5</continentIndex>
  </country>
  <country>
    <name>China</name>
    <population>1265530000</population>
    <continentIndex>1</continentIndex>
  </country>
  <country>
    <name>Chile</name>
    <population>14677000</population>
    <continentIndex>3</continentIndex>
  </country>
. . .
</countries>
```

Notice that each `<country>` entry has a `<continentIndex>` entry, which is a numeric value to represent the continent. Using the drop-down form field, you can create an index in your template that will cross-reference the `<continentIndex>` value to the actual continent name. You can then display the name in your published report.

To create the index for the continent example:

1. Position the cursor in your template where you want the value from the drop-down list to display, and select the Drop-Down Form Field from the Forms tool bar (shown in the following figure).

2. Right-click the field to display the Drop-Down Form Field Options dialog.

3. Add each value to the Drop-down item field and the click Add to add it to the Items in drop-down list group. The values will be indexed starting from one for the first, and so on. For example, the list of continents will be stored as follows:
4. Now use the Help Text box to enter the XML element name that will hold the index for the drop-down field values.

For this example, enter

```xml
<?continentIndex?>
```

The following figure shows the **Drop-Down Form Field Options** dialogs for this example:

Using the check box and drop-down list features, you can create a report to display population data with check boxes to demonstrate figures that reach a certain limit. An example is shown in the following figure:

<table>
<thead>
<tr>
<th>Country</th>
<th>Population</th>
<th>more than 10M?</th>
<th>Continent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chad</td>
<td>7,360,000</td>
<td></td>
<td>Africa</td>
</tr>
<tr>
<td>China</td>
<td>1,265,530,000</td>
<td></td>
<td>Asia</td>
</tr>
<tr>
<td>Chile</td>
<td>14,677,000</td>
<td>✗</td>
<td>South America</td>
</tr>
<tr>
<td>Sweden</td>
<td>8,987,000</td>
<td></td>
<td>Europe</td>
</tr>
<tr>
<td>United States</td>
<td>270,312,000</td>
<td>✗</td>
<td>North America</td>
</tr>
<tr>
<td>New Zealand</td>
<td>3,625,000</td>
<td></td>
<td>Australia</td>
</tr>
</tbody>
</table>

The template to create this report is shown in the next figure:
where the fields have the following values:

<table>
<thead>
<tr>
<th>Field</th>
<th>Form Field Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FE</td>
<td>&lt;?for-each:country?&gt;</td>
<td>Begins the <em>country</em> repeating group.</td>
</tr>
<tr>
<td>China</td>
<td>&lt;?name?&gt;</td>
<td>Placeholder for the <em>name</em> element.</td>
</tr>
<tr>
<td>1,000,000</td>
<td>&lt;?population?&gt;</td>
<td>Placeholder for the <em>population</em> element.</td>
</tr>
<tr>
<td>(check box)</td>
<td>&lt;?population&gt;1000000?&gt;</td>
<td>Establishes the condition for the check box. If the value for the <em>population</em> element is greater than 1,000,000, the check box will display as checked.</td>
</tr>
<tr>
<td>Asia</td>
<td>&lt;?continentIndex?&gt;</td>
<td>The drop-down form field for the <em>continentIndex</em> element. See the preceding description for its contents. At runtime, the value of the XML element is replaced with the value it is cross-referenced to in the drop-down form field.</td>
</tr>
<tr>
<td>EFE</td>
<td>&lt;?end for-each?&gt;</td>
<td>Ends the <em>country</em> group.</td>
</tr>
</tbody>
</table>

### Conditional Formatting

Conditional formatting occurs when a formatting element appears only when a certain condition is met. XML Publisher supports the usage of simple "if" statements, as well as more complex "choose" expressions.

The conditional formatting that you specify can be XSL or XSL:FO code, or you can specify actual RTF objects such as a table or data. For example, you can specify that if reported numbers reach a certain threshold, they will display shaded in red. Or, you can use this feature to hide table columns or rows depending on the incoming XML data.

### If Statements

Use an if statement to define a simple condition; for example, if a data field is a specific value.

1. Insert the following syntax to designate the beginning of the conditional area.

   ```xml
   <?if:condition?>
   ```

2. Insert the following syntax at the end of the conditional area: `<?end if?>`.

For example, to set up the Payables Invoice Register to display invoices only when the Supplier name is "Company A", insert the syntax `<?if:VENDOR_NAME='COMPANY A'>` before the Supplier field on the template.

Enter the `<?end if?>` tag after the invoices table.

This example is displayed in the figure below. Note that you can insert the syntax in form fields, or directly into the template.
If Statements in Boilerplate Text

Assume you want to incorporate an "if" statement into the following free-form text:

The program was (not) successful.

You only want the "not" to display if the value of an XML tag called <SUCCESS> equals "N".

To achieve this requirement, you must use the XML Publisher context command to place the if statement into the inline sequence rather than into the block (the default placement).

Note: For more information on context commands, see Using Context Commands, page 2-107.

For example, if you construct the code as follows:

The program was <?if:SUCCESS='N'>not<?end if?> successful.

The following undesirable result will occur:

The program was
not
successful.
because XML Publisher applies the instructions to the block by default. To specify that the if statement should be inserted into the inline sequence, enter the following:

The program was <?if@inlines:SUCCESS='N'>not<?end if?> successful.
This construction will result in the following display:

The program was successful.
If SUCCESS does not equal 'N';
or
The program was not successful.
If SUCCESS equals 'N'.
If-then-Else Statements

XML Publisher supports the common programming construct "if-then-else". This is extremely useful when you need to test a condition and conditionally show a result. For example:

```plaintext
IF X=0 THEN
  Y=2
ELSE
  Y=3
END IF
```

You can also nest these statements as follows:

```plaintext
IF X=0 THEN
  Y=2
ELSE
  IF X=1 THEN
    Y=10
  ELSE
    Y=100
  END IF
END IF
```

Use the following syntax to construct an if-then-else statement in your RTF template:

```xml
<?xdofx:if element_condition then result1 else result2 end if?>
```

For example, the following statement tests the AMOUNT element value. If the value is greater than 1000, show the word "Higher"; if it is less than 1000, show the word "Lower"; if it is equal to 1000, show "Equal":

```xml
<?xdofx:if AMOUNT > 1000 then 'Higher'
  else
    if AMOUNT < 1000 then 'Lower'
    else
      'Equal'
  end if?>
```

Choose Statements

Use the `choose`, `when`, and `otherwise` elements to express multiple conditional tests. If certain conditions are met in the incoming XML data then specific sections of the template will be rendered. This is a very powerful feature of the RTF template. In regular XSL programming, if a condition is met in the `choose` command then further XSL code is executed. In the template, however, you can actually use visual widgets in the conditional flow (in the following example, a table).

Use the following syntax for these elements:

```xml
<?choose:?>
<?when:expression?>
<?otherwise?>
```

"Choose" Conditional Formatting Example

This example shows a `choose` expression in which the display of a row of data depends on the value of the fields EXEMPT_FLAG and POSTED_FLAG. When the EXEMPT_FLAG equals "^", the row of data will render light gray. When POSTED_FLAG equals "*", the row of data will render shaded dark gray. Otherwise, the row of data will render with no shading.
In the following figure, the form field default text is displayed. The form field help text entries are shown in the table following the example.

<table>
<thead>
<tr>
<th>Default Text Entry in Example Form Field</th>
<th>Help Text Entry in Form Field</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="">Grp:VAT</a></td>
<td>&lt;?for-each:VAT?&gt;</td>
</tr>
<tr>
<td>&lt;Choose</td>
<td>&lt;?choose?&gt;</td>
</tr>
<tr>
<td>&lt;When EXEMPT_FLAG=&quot;^^&quot;&gt;</td>
<td>&lt;?When EXEMPT_FLAG=&quot;^^&quot;&gt;</td>
</tr>
<tr>
<td>End When&gt;</td>
<td>&lt;?end When?&gt;</td>
</tr>
<tr>
<td>&lt;When EXEMPT_FLAG=&quot;^^&quot;&gt;</td>
<td>&lt;?When EXEMPT_FLAG=&quot;^^&quot;&gt;</td>
</tr>
<tr>
<td>End When&gt;</td>
<td>&lt;?end When?&gt;</td>
</tr>
</tbody>
</table>
Column Formatting

You can conditionally show and hide columns of data in your document output. The following example demonstrates how to set up a table so that a column is only displayed based on the value of an element attribute.

This example will show a report of a price list, represented by the following XML:

```xml
<items type="PUBLIC"> <!-- can be marked ‘PRIVATE’ -->
  <item>
    <name>Plasma TV</name>
    <quantity>10</quantity>
    <price>4000</price>
  </item>
  <item>
    <name>DVD Player</name>
    <quantity>3</quantity>
    <price>300</price>
  </item>
  <item>
    <name>VCR</name>
    <quantity>20</quantity>
    <price>200</price>
  </item>
  <item>
    <name>Receiver</name>
    <quantity>22</quantity>
    <price>350</price>
  </item>
</items>
```

Notice the `type` attribute associated with the `items` element. In this XML it is marked as "PUBLIC" meaning the list is a public list rather than a "PRIVATE" list. For the "public" version of the list we do not want to show the quantity column in the output, but we want to develop only one template for both versions based on the list type.

The following figure is a simple template that will conditionally show or hide the quantity column:

<table>
<thead>
<tr>
<th>Name</th>
<th>IF Quantity=20 THEN end-if</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plasma TV</td>
<td></td>
<td>4000</td>
</tr>
<tr>
<td>DVD Player</td>
<td></td>
<td>300</td>
</tr>
<tr>
<td>VCR</td>
<td></td>
<td>200</td>
</tr>
<tr>
<td>Receiver</td>
<td></td>
<td>350</td>
</tr>
</tbody>
</table>

The following table shows the entries made in the template for the example:
<table>
<thead>
<tr>
<th>Default Text</th>
<th>Form Field Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grp:Item</td>
<td>&lt;?for-each:item?&gt;</td>
<td>Holds the opening for-each loop for the item element.</td>
</tr>
<tr>
<td>Plasma TV</td>
<td>&lt;?name?&gt;</td>
<td>The placeholder for the name element from the XML file.</td>
</tr>
<tr>
<td>IF</td>
<td>&lt;?if@column:/items/@type=&quot;PRIVATE&quot;?&gt;</td>
<td>The opening of the if statement to test for the attribute value &quot;PRIVATE&quot;. Note that this syntax uses an XPath expression to navigate back to the &quot;items&quot; level of the XML to test the attribute. For more information about using XPath in your templates, see XPath Overview, page 2-104.</td>
</tr>
<tr>
<td>Quantity</td>
<td>N/A</td>
<td>Boilerplate heading</td>
</tr>
<tr>
<td>end-if</td>
<td>&lt;?end if?&gt;</td>
<td>Ends the if statement.</td>
</tr>
<tr>
<td>20</td>
<td>&lt;?if@column:/items/@type=&quot;PRIVATE&quot;?&gt;&lt;?quantity?&gt;&lt;?end if?&gt;</td>
<td>The placeholder for the quantity element surrounded by the &quot;if&quot; statement.</td>
</tr>
<tr>
<td>1,000.00</td>
<td>&lt;?price?&gt;</td>
<td>The placeholder for the price element.</td>
</tr>
<tr>
<td>end grp</td>
<td>&lt;?end for-each?&gt;</td>
<td>Closing tag of the for-each loop.</td>
</tr>
</tbody>
</table>

The conditional column syntax is the "if" statement syntax with the addition of the @column clause. It is the @column clause that instructs XML Publisher to hide or show the column based on the outcome of the if statement.

If you did not include the @column the data would not display in your report as a result of the if statement, but the column still would because you had drawn it in your template.

**Note:** The @column clause is an example of a context command. For more information, see Using Context Commands, page 2-107.

The example will render the output shown in the following figure:

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plasma TV</td>
<td>4,000.00</td>
</tr>
<tr>
<td>DVD Player</td>
<td>300.00</td>
</tr>
<tr>
<td>VCR</td>
<td>200.00</td>
</tr>
<tr>
<td>Receiver</td>
<td>350.00</td>
</tr>
</tbody>
</table>

If the same XML data contained the type attribute set to "PRIVATE" the following output would be rendered from the same template:
Row Formatting

XML Publisher allows you to specify formatting conditions as the row-level of a table. Examples of row-level formatting are:

- Highlighting a row when the data meets a certain threshold.
- Alternating background colors of rows to ease readability of reports.
- Showing only rows that meet a specific condition.

Conditionally Displaying a Row

To display only rows that meet a certain condition, insert the `<if:condition?> <?end if?>` tags at the beginning and end of the row, within the for-each tags for the group. This is demonstrated in the following sample template.

<table>
<thead>
<tr>
<th>Industry</th>
<th>Year</th>
<th>Month</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>for-each SALE if big INDUSTRY</td>
<td>YEAR</td>
<td>MONTH</td>
<td>SALES end if end SALE</td>
</tr>
</tbody>
</table>

Note the following fields from the sample figure:

<table>
<thead>
<tr>
<th>Default Text Entry</th>
<th>Form Field Help Text</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>for-each SALE</td>
<td>&lt;?for-each:SALE?&gt;</td>
<td>Opens the for-each loop to repeat the data belonging to the SALE group.</td>
</tr>
<tr>
<td>if big</td>
<td>&lt;?if:SALES&gt;5000?&gt;</td>
<td>If statement to display the row only if the element SALES has a value greater than 5000.</td>
</tr>
<tr>
<td>INDUSTRY</td>
<td>&lt;?INDUSTRY?&gt;</td>
<td>Data field</td>
</tr>
<tr>
<td>YEAR</td>
<td>&lt;?YEAR?&gt;</td>
<td>Data field</td>
</tr>
<tr>
<td>MONTH</td>
<td>&lt;?MONTH?&gt;</td>
<td>Data field</td>
</tr>
<tr>
<td>SALES end if</td>
<td>&lt;?end if?&gt;</td>
<td>Closes the if statement.</td>
</tr>
<tr>
<td>end SALE</td>
<td>&lt;?end for-each?&gt;</td>
<td>Closes the SALE loop.</td>
</tr>
</tbody>
</table>

Conditionally Highlighting a Row

This example demonstrates how to set a background color on every other row. The template to create this effect is shown in the following figure:

<table>
<thead>
<tr>
<th>Industry</th>
<th>Year</th>
<th>Month</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>for-each SALE format, INDUSTRY</td>
<td>YEAR</td>
<td>MONTH</td>
<td>SALES end SALE</td>
</tr>
</tbody>
</table>
The following table shows values of the form fields in the template:

<table>
<thead>
<tr>
<th>Default Text Entry</th>
<th>Form Field Help Text</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>for-each SALE</td>
<td>&lt;?for-each:SALE?&gt;</td>
<td>Defines the opening of the for-each loop for the SALE group.</td>
</tr>
</tbody>
</table>
| format;            | <?if @row:position() mod 2=0?>
|                    | <xsl:attribute name="background-color" xdofo:ctx="incontext">lightgray</xsl:attribute><?end if?> |
|                    |                       | For each alternate row, the background color attribute is set to gray for the row. |
| INDUSTRY           | <?INDUSTRY?>          | Data field |
| YEAR               | <?YEAR?>              | Data field |
| MONTH              | <?MONTH?>             | Data field |
| SALES              | <?SALES?>             | Data field |
| end SALE           | <?end for-each?>      | Closes the SALE for-each loop. |

In the preceding example, note the "format;" field. It contains an if statement with a "row" context (@row). This sets the context of the if statement to apply to the current row. If the condition is true, then the <xsl:attribute> for the background color of the row will be set to light gray. This will result in the following output:

<table>
<thead>
<tr>
<th>Industry</th>
<th>Year</th>
<th>Month</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oil</td>
<td>2000</td>
<td>Jan</td>
<td>100,000</td>
</tr>
<tr>
<td>Automotive</td>
<td>2000</td>
<td>Jan</td>
<td>200,000</td>
</tr>
<tr>
<td>Groceries</td>
<td>2000</td>
<td>Jan</td>
<td>50,000</td>
</tr>
</tbody>
</table>

**Note:** For more information about context commands, see Using Context Commands., page 2-107

**Cell Highlighting**

The following example demonstrates how to conditionally highlight a cell based on a value in the XML file.

For this example we will use the following XML:
The template lists the accounts and their credit and debit values. In the final report we want to highlight in red any cell whose value is greater than 1000. The template for this is shown in the following graphic:

<table>
<thead>
<tr>
<th>Account</th>
<th>Debit</th>
<th>Credit</th>
</tr>
</thead>
<tbody>
<tr>
<td>FE:Account1-232-4444</td>
<td>100.00</td>
<td>CH2100.00</td>
</tr>
</tbody>
</table>

The field definitions for the template are shown in the following table:

<table>
<thead>
<tr>
<th>Default Text Entry</th>
<th>Form Field Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FE:Account</td>
<td>&lt;?for-each:account?&gt;</td>
<td>Opens the for-each-loop for the element account.</td>
</tr>
<tr>
<td>1-232-4444</td>
<td>&lt;?number?&gt;</td>
<td>The placeholder for the number element from the XML file.</td>
</tr>
<tr>
<td>CH1</td>
<td>&lt;?if:debit&gt;1000?&gt;&lt;xsl:attribute xdofo:ctx=&quot;block&quot; name=&quot;background-color&quot;&gt;red&lt;/xsl:attribute&gt;&lt;?end if?&gt;</td>
<td>This field holds the code to highlight the cell red if the debit amount is greater than 1000.</td>
</tr>
<tr>
<td>100.00</td>
<td>&lt;?debit?&gt;</td>
<td>The placeholder for the debit element.</td>
</tr>
<tr>
<td>CH2</td>
<td>&lt;?if:credit&gt;1000?&gt;&lt;xsl:attribute xdofo:ctx=&quot;block&quot; name=&quot;background-color&quot;&gt;red&lt;/xsl:attribute&gt;&lt;?end if?&gt;</td>
<td>This field holds the code to highlight the cell red if the credit amount is greater than 1000.</td>
</tr>
<tr>
<td>100.00</td>
<td>&lt;?credit?&gt;</td>
<td>The placeholder for the credit element.</td>
</tr>
<tr>
<td>EFE</td>
<td>&lt;?end for-each?&gt;</td>
<td>Closes the for-each loop.</td>
</tr>
</tbody>
</table>
The code to highlight the debit column as shown in the table is:

```xml
<?if:debit>1000?>
  <xsl:attribute
      xdofo:ctx="block" name="background-color">red
  </xsl:attribute>
<?end if?>
```

The "if" statement is testing if the debit value is greater than 1000. If it is, then the next lines are invoked. Notice that the example embeds native XSL code inside the "if" statement.

The "attribute" element allows you to modify properties in the XSL.

The xdo:ctx component is an XML Publisher feature that allows you to adjust XSL attributes at any level in the template. In this case, the background color attribute is changed to red.

To change the color attribute, you can use either the standard HTML names (for example, red, white, green) or you can use the hexadecimal color definition (for example, #FFFFF).

The output from this template is displayed in the following figure:

<table>
<thead>
<tr>
<th>Account</th>
<th>Debit</th>
<th>Credit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-100-3333</td>
<td>100.00</td>
<td>300.00</td>
</tr>
<tr>
<td>1-101-2532</td>
<td>220.00</td>
<td>30.00</td>
</tr>
<tr>
<td>1-130-3343</td>
<td>240.00</td>
<td></td>
</tr>
<tr>
<td>1-153-3033</td>
<td>300.00</td>
<td>300.00</td>
</tr>
</tbody>
</table>

**Page-Level Calculations**

**Displaying Page Totals**

XML Publisher allows you to display calculated page totals in your report. Because the page is not created until publishing time, the totaling function must be executed by the formatting engine.

*Note:* Page totaling is performed in the PDF-formatting layer. Therefore this feature is not available for other outputs types: HTML, RTF, Excel.

Because the page total field does not exist in the XML input data, you must define a variable to hold the value. When you define the variable, you associate it with the element from the XML file that is to be totaled for the page. Once you define total fields, you can also perform additional functions on the data in those fields.

To declare the variable that is to hold your page total, insert the following syntax immediately following the placeholder for the element that is to be totaled:

```xml
<?add-page-total:TotalFieldName;‘element’?>
```

where

*TotalFieldName* is the name you assign to your total (to reference later) and

*‘element’* is the XML element field to be totaled.

You can add this syntax to as many fields as you want to total.

Then when you want to display the total field, enter the following syntax:
The following example shows how to set up page total fields in a template to display total credits and debits that have displayed on the page, and then calculate the net of the two fields.

This example uses the following XML:

```xml
<balance_sheet>
  <transaction>
    <debit>100</debit>
    <credit>90</credit>
  </transaction>
  <transaction>
    <debit>110</debit>
    <credit>80</credit>
  </transaction>
  ...
</balance_sheet>
```

The following figure shows the table to insert in the template to hold the values:

<table>
<thead>
<tr>
<th>Debit</th>
<th>Credit</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.00</td>
<td>90.00</td>
</tr>
</tbody>
</table>

The following table shows the form field entries made in the template for the example table:
<table>
<thead>
<tr>
<th>Default Text Entry</th>
<th>Form Field Help Text Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FE</td>
<td>&lt;?for-each:transaction?&gt;</td>
<td>This field defines the opening &quot;for-each&quot; loop for the transaction group.</td>
</tr>
<tr>
<td>100.00</td>
<td>&lt;?debit?&gt;<a href="">add-page-total:dt;’debit’?</a></td>
<td>This field is the placeholder for the debit element from the XML file. Because we want to total this field by page, the page total declaration syntax is added. The field defined to hold the total for the debit element is dt.</td>
</tr>
<tr>
<td>90.00</td>
<td>&lt;?credit?&gt;<a href="">add-page-total:ct;’credit’?</a></td>
<td>This field is the placeholder for the credit element from the XML file. Because we want to total this field by page, the page total declaration syntax is added. The field defined to hold the total for the credit element is ct.</td>
</tr>
<tr>
<td>Net</td>
<td>&lt;add-page-total:net;’debit - credit’?&gt;</td>
<td>Creates a net page total by subtracting the credit values from the debit values.</td>
</tr>
<tr>
<td>EFE</td>
<td>&lt;?end for-each?&gt;</td>
<td>Closes the for-each loop.</td>
</tr>
</tbody>
</table>

Note that on the field defined as "net" we are actually carrying out a calculation on the values of the credit and debit elements.

Now that you have declared the page total fields, you can insert a field in your template where you want the page totals to appear. Reference the calculated fields using the names you supplied (in the example, ct and dt). The syntax to display the page totals is as follows:

For example, to display the debit page total, you could enter the following:

```
<?show-page-total:dt;$#,##0.00; ($#,##0.00)’?>
```

Therefore to complete the example, place the following at the bottom of the template page, or in the footer:

Page Total Debit: <?show-page-total:dt;$#,##0.00; ($#,##0.00)’?>

Page Total Credit: <?show-page-total:ct;$#,##0.00; ($#,##0.00)’?>

Page Total Balance: <?show-page-total:net;$#,##0.00; ($#,##0.00)’?>

The output for this report is shown in the following graphic:
Note that this page totaling function will only work if your source XML has raw numeric values. The numbers must not be preformatted.

**Brought Forward/Carried Forward Totals**

Many reports require that a page total be maintained throughout the report output and be displayed at the beginning and end of each page. These totals are known as "brought forward/carried forward" totals.

**Note:** The totaling for the brought forward and carried forward fields is performed in the PDF-formatting layer. Therefore this feature is not available for other outputs types: HTML, RTF, Excel.

An example is displayed in the following figure:

At the end of the first page, the page total for the Amount element is displayed as the Carried Forward total. At the top of the second page, this value is displayed as the Brought Forward total from the previous page. At the bottom of the second page, the brought forward value plus the total for that page is calculated and displayed as the new Carried Forward value, and this continues throughout the report.

This functionality is an extension of the Page Totals, page 2-64 feature. The following example walks through the syntax and setup required to display the brought forward and carried forward totals in your published report.

Assume you have the following XML:
The following sample template creates the invoice table and declares a placeholder that will hold your page total:

```
<xml version="1.0" encoding="WINDOWS-1252"/>
</INVOICES>

The fields in the template have the following values:

<table>
<thead>
<tr>
<th>Field</th>
<th>Form Field Help Text Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init PTs</td>
<td>&lt;?init-page-total: InvAmt?&gt;</td>
<td>Declares &quot;InvAmt&quot; as the placeholder that will hold the page total.</td>
</tr>
<tr>
<td>FE</td>
<td>&lt;?for-each:INVOICE?&gt;</td>
<td>Begins the INVOICE group.</td>
</tr>
<tr>
<td>100.00</td>
<td>&lt;?INVAMT?&gt;</td>
<td>Placeholder for the Invoice Amount tag.</td>
</tr>
<tr>
<td>InvAmt</td>
<td>&lt;?add-page-total:InvAmt;INVAMT?&gt;</td>
<td>Assigns the &quot;InvAmt&quot; page total object to the INVAMT element in the data.</td>
</tr>
<tr>
<td>EFE</td>
<td>&lt;?end for-each?&gt;</td>
<td>Closes the INVOICE group.</td>
</tr>
<tr>
<td>End PTs</td>
<td>&lt;?end-page-total:InvAmt?&gt;</td>
<td>Closes the &quot;InvAmt&quot; page total.</td>
</tr>
</tbody>
</table>
```

To display the brought forward total at the top of each page (except the first), use the following syntax:

```
```
The following table describes the elements comprising the brought forward syntax:

<table>
<thead>
<tr>
<th>Code Element</th>
<th>Description and Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>inline-total</td>
<td>This element has two properties:</td>
</tr>
<tr>
<td></td>
<td>• name - name of the variable you declared for the field.</td>
</tr>
<tr>
<td></td>
<td>• display-condition - sets the display condition. This is an optional property that</td>
</tr>
<tr>
<td></td>
<td>takes one of the following values:</td>
</tr>
<tr>
<td></td>
<td>• first - the contents appear only on the first page</td>
</tr>
<tr>
<td></td>
<td>• last - the contents appear only on the last page</td>
</tr>
<tr>
<td></td>
<td>• exceptfirst - contents appear on all pages except first</td>
</tr>
<tr>
<td></td>
<td>• exceptlast - contents appear on all pages except last</td>
</tr>
<tr>
<td></td>
<td>• everytime - (default) contents appear on every page</td>
</tr>
<tr>
<td></td>
<td>In this example, display-condition is set to &quot;exceptfirst&quot; to prevent the value</td>
</tr>
<tr>
<td></td>
<td>from appearing on the first page where the value would be zero.</td>
</tr>
<tr>
<td></td>
<td>Brought Forward:</td>
</tr>
<tr>
<td>show-brought-forward</td>
<td>This string is optional and will display as the field name on the report.</td>
</tr>
<tr>
<td></td>
<td>Shows the value on the page. It has the following two properties:</td>
</tr>
<tr>
<td></td>
<td>• name - the name of the field to show. In this case, &quot;InvAmt&quot;. This property is</td>
</tr>
<tr>
<td></td>
<td>mandatory.</td>
</tr>
<tr>
<td></td>
<td>• format - the Oracle number format to apply to the value at runtime. This property</td>
</tr>
<tr>
<td></td>
<td>is optional. For more information about number formats, see Number and Date</td>
</tr>
<tr>
<td></td>
<td>Formatting, page 2-91.</td>
</tr>
</tbody>
</table>

Insert the brought forward object at the top of the template where you want the brought forward total to display. If you place it in the body of the template, you can insert the syntax in a form field.

If you want the brought forward total to display in the header, you must insert the full code string into the header because Microsoft Word does not support form fields in the header or footer regions. However, you can alternatively use the start body/end body syntax which allows you to define what the body area of the report will be. XML Publisher will recognize any content above the defined body area as header content, and any content below as the footer. This allows you to use form fields. See Multiple or Complex Headers and Footers, page 2-13 for details.

Place the carried forward object at the bottom of your template where you want the total to display. The carried forward object for our example is as follows:
Carried Forward:

Note the following differences with the brought-forward object:

- The display-condition is set to exceptlast so that the carried forward total will display on every page except the last page.
- The display string is "Carried Forward".
- The show-carry-forward element is used to show the carried forward value. It has the same properties as brought-carried-forward, described above.

You are not limited to a single value in your template, you can create multiple brought forward/carried forward objects in your template pointing to various numeric elements in your data.

Running Totals

Example

The variable functionality (see Using Variables, page 2-79) can be used to add a running total to your invoice listing report. This example assumes the following XML structure:

```xml
<?xml version="1.0" encoding="WINDOWS-1252"?>
<INVOICES>
  <INVOICE>
    <INVNUM>10001-1</INVNUM>
    <INVDATE>1-Jan-2005</INVDATE>
    <INVAUTH>100</INVAUTH>
  </INVOICE>
  <INVOICE>
    <INVNUM>10001-2</INVNUM>
    <INVDATE>10-Jan-2005</INVDATE>
    <INVAUTH>200</INVAUTH>
  </INVOICE>
  <INVOICE>
    <INVNUM>10001-1</INVNUM>
    <INVDATE>11-Jan-2005</INVDATE>
    <INVAUTH>150</INVAUTH>
  </INVOICE>
</INVOICES>
```

Using this XML, we want to create the report that contains running totals as shown in the following figure:

<table>
<thead>
<tr>
<th>Invoice Number</th>
<th>Invoice Date</th>
<th>Amount</th>
<th>Running Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000-1</td>
<td>1-Jan-2005</td>
<td>100.00</td>
<td>100.00</td>
</tr>
<tr>
<td>1000-2</td>
<td>10-Jan-2005</td>
<td>200.00</td>
<td>300.00</td>
</tr>
<tr>
<td>1000-3</td>
<td>11-Jan-2005</td>
<td>150.00</td>
<td>450.00</td>
</tr>
</tbody>
</table>
To create the Running Total field, define a variable to track the total and initialize it to 0. The template is shown in the following figure:

<table>
<thead>
<tr>
<th>Invoice Number</th>
<th>Invoice Date</th>
<th>Amount</th>
<th>Running Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>FE10001-1</td>
<td>1-Jan-2005</td>
<td>100.00</td>
<td>100.00</td>
</tr>
</tbody>
</table>

The values for the form fields in the template are shown in the following table:

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RtotalVar</td>
<td>&lt;?xdoxslt:set_variable($_XDOCTX, 'RtotalVar', 0)?&gt;</td>
<td>Declares the &quot;RtotalVar&quot; variable and initializes it to 0.</td>
</tr>
<tr>
<td>FE</td>
<td>&lt;?for-each:INVOICE?&gt;</td>
<td>Starts the Invoice group.</td>
</tr>
<tr>
<td>10001-1</td>
<td>&lt;?INVNUM?&gt;</td>
<td>Invoice Number tag</td>
</tr>
<tr>
<td>1-Jan-2005</td>
<td>&lt;?INVDATE?&gt;</td>
<td>Invoice Date tag</td>
</tr>
<tr>
<td>100.00</td>
<td>&lt;?xdoxslt:set_variable($_XDOCTX, 'RtotalVar', xdoxslt:get_variable($_XDOCTX, 'RtotalVar') + INVAMT)?&gt;</td>
<td>Sets the value of RTotalVar to the current value plus the new Invoice Amount.</td>
</tr>
<tr>
<td></td>
<td>xdoxslt:get_variable($_XDOCTX, 'RTotalVar')?&gt;</td>
<td>Retrieves the RTotalVar value for display.</td>
</tr>
<tr>
<td>EFE</td>
<td>&lt;?end for-each?&gt;</td>
<td>Ends the INVOICE group.</td>
</tr>
</tbody>
</table>

**Data Handling**

**Sorting**

You can sort a group by any element within the group. Insert the following syntax within the group tags:

<?sort:element name?>

For example, to sort the Payables Invoice Register (shown at the beginning of this chapter) by Supplier (VENDOR_NAME), enter the following after the <?for-each:G_VENDOR_NAME?> tag:

<?sort:VENDOR_NAME?>

To sort a group by multiple fields, just insert the sort syntax after the primary sort field. To sort by Supplier and then by Invoice Number, enter the following


**Checking for Nulls**

Within your XML data there are three possible scenarios for the value of an element:

- The element is present in the XML data, and it has a value
- The element is present in the XML data, but it does not have a value
The element is not present in the XML data, and therefore there is no value

In your report layout, you may want to specify a different behavior depending on the presence of the element and its value. The following examples show how to check for each of these conditions using an "if" statement. The syntax can also be used in other conditional formatting constructs.

- To define behavior when the element is present and the value is not null, use the following:
  
  
  ```xml
  <?if:element_name!=?> desired behavior when the element exists and is not null <?end if?>
  
  - To define behavior when the element is present, but is null, use the following:

  ```xml
  <?if:element_name and element_name="?> desired behavior when the element exists but value is null <?end if?>
  
  - To define behavior when the element is not present, use the following:

  ```xml
  <?if:not(element_name)?> desired behavior when the element is not present <?end if?>
  
  Regrouping the XML Data

  The RTF template supports the XSL 2.0 for-each-group standard that allows you to regroup XML data into hierarchies that are not present in the original data. With this feature, your template does not have to follow the hierarchy of the source XML file. You are therefore no longer limited by the structure of your data source.

  XML Sample

  To demonstrate the for-each-group standard, the following XML data sample of a CD catalog listing will be regrouped in a template:
Using the regrouping syntax, you can create a report of this data that groups the CDs by country and then by year. You are not limited by the data structure presented.

### Regrouping Syntax

To regroup the data, use the following syntax:

```
<?for-each-group: BASE-GROUP; GROUPING-ELEMENT?>
```

For example, to regroup the CD listing by COUNTRY, enter the following in your template:

```
<?for-each-group: CD; COUNTRY?>
```

The elements that were at the same hierarchy level as COUNTRY are now children of COUNTRY. You can then refer to the elements of the group to display the values desired.

To establish nested groupings within the already defined group, use the following syntax:

```
<?for-each:current-group(); GROUPING-ELEMENT?>
```

For example, after declaring the CD grouping by COUNTRY, you can then further group by YEAR within COUNTRY as follows:

```
<?for-each:current-group(); YEAR?>
```

At runtime, XML Publisher will loop through the occurrences of the new groupings, displaying the fields that you defined in your template.

**Note:** This syntax is a simplification of the XSL for-each-group syntax. If you choose not to use the simplified syntax above, you can use the XSL
syntax as shown below. The XSL syntax can only be used within a form field of the template.

```xml
<xsl:for-each-group
    select=expression
    group-by="string expression"
    group-adjacent="string expression"
    group-starting-with=pattern>
    <!--Content: (xsl:sort*, content-constructor) -->
</xsl:for-each-group>
```

**Template Example**

The following figure shows a template that displays the CDs by Country, then Year, and lists the details for each CD:

<table>
<thead>
<tr>
<th>Group by Country</th>
<th>Country: USA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group by Year</td>
<td>Year: 2000</td>
</tr>
<tr>
<td>Title</td>
<td>Artist</td>
</tr>
<tr>
<td>Group Details: CD</td>
<td>John Doe</td>
</tr>
</tbody>
</table>

The following table shows the XML Publisher syntax entries made in the form fields of the preceding template:
<table>
<thead>
<tr>
<th>Default Text Entry</th>
<th>Form Field Help Text Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group by Country</td>
<td><code>&lt;?for-each-group: CD;COUNTRY?&gt;</code></td>
<td>The <code>&lt;?for-each-group: CD;COUNTRY?&gt;</code> tag declares the new group. It regroups the existing CD group by the COUNTRY element.</td>
</tr>
<tr>
<td>USA</td>
<td><code>&lt;?COUNTRY?&gt;</code></td>
<td>Placeholder to display the data value of the COUNTRY tag.</td>
</tr>
<tr>
<td>Group by Year</td>
<td><code>&lt;?for-each-group: current-group();YEAR?&gt;</code></td>
<td>The <code>&lt;?for-each-group: current-group();YEAR?&gt;</code> tag regroups the current group (that is, COUNTRY), by the YEAR element.</td>
</tr>
<tr>
<td>2000</td>
<td><code>&lt;?YEAR?&gt;</code></td>
<td>Placeholder to display the data value of the YEAR tag.</td>
</tr>
<tr>
<td>Group: Details</td>
<td><code>&lt;?for-each:current-group()?&gt;</code></td>
<td>Once the data is grouped by COUNTRY and then by YEAR, the <code>&lt;?for-each:current-group()?&gt;</code> command is used to loop through the elements of the current group (that is, YEAR) and render the data values (TITLE, ARTIST, and PRICE) in the table.</td>
</tr>
<tr>
<td>My CD</td>
<td><code>&lt;?TITLE?&gt;</code></td>
<td>Placeholder to display the data value of the TITLE tag.</td>
</tr>
<tr>
<td>John Doe</td>
<td><code>&lt;?ARTIST?&gt;</code></td>
<td>Placeholder to display the data value of the ARTIST tag.</td>
</tr>
<tr>
<td>1.00</td>
<td><code>&lt;?PRICE?&gt;</code></td>
<td>Placeholder to display the data value of the PRICE tag.</td>
</tr>
<tr>
<td>End Group</td>
<td><code>&lt;?end for-each?&gt;</code></td>
<td>Closes out the <code>&lt;?for-each: current-group()?&gt;</code> tag.</td>
</tr>
<tr>
<td>End Group by Year</td>
<td><code>&lt;?end for-each-group?&gt;</code></td>
<td>Closes out the <code>&lt;?for-each-group:current-group();YEAR?&gt;</code> tag.</td>
</tr>
</tbody>
</table>

This template produces the following output when merged with the XML file:
Regrouping by an Expression

Regrouping by an expression allows you to apply a function or command to a data element, and then group the data by the returned result.

To use this feature, state the expression within the regrouping syntax as follows:

```xml
<?for-each:BASE-GROUP; GROUPING-EXPRESSION?>
```

Example

To demonstrate this feature, an XML data sample that simply contains average temperatures per month will be used as input to a template that calculates the number of months having an average temperature within a certain range.

The following XML sample is composed of `<temp>` groups. Each `<temp>` group contains a `<month>` element and a `<degree>` element, which contains the average temperature for that month:
You want to display this data in a format showing temperature ranges and a count of the months that have an average temperature to satisfy those ranges, as follows:
Annual Temperature Summary

<table>
<thead>
<tr>
<th>Range</th>
<th>Number of Months</th>
</tr>
</thead>
<tbody>
<tr>
<td>0°F to 10°F</td>
<td>1 Month(s)</td>
</tr>
<tr>
<td>10°F to 20°F</td>
<td>4 Month(s)</td>
</tr>
<tr>
<td>20°F to 30°F</td>
<td>3 Month(s)</td>
</tr>
<tr>
<td>30°F to 40°F</td>
<td>4 Month(s)</td>
</tr>
</tbody>
</table>

Using the for-each-group command you can apply an expression to the `<degree>` element that will enable you to group the temperatures by increments of 10 degrees. You can then display a count of the members of each grouping, which will be the number of months having an average temperature that falls within each range.

The template to create the above report is shown in the following figure:

### Annual Temperature Summary

<table>
<thead>
<tr>
<th>Range</th>
<th>Number of Months</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group by TmpRng</td>
<td>Months Month(s)</td>
</tr>
<tr>
<td>Range</td>
<td>End TmpRng</td>
</tr>
</tbody>
</table>

The following table shows the form field entries made in the template:

<table>
<thead>
<tr>
<th>Default Text Entry</th>
<th>Form Field Help Text Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group by TmpRng</td>
<td><code>&lt;?for-each-group:temp;floor(degree div 10)&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;?sort:floor(degree div 10)?&gt;</code></td>
</tr>
<tr>
<td>Range</td>
<td><code>&lt;?concat(floor(degree div 10)*10,' F to ',floor(degree div 10)*10+10, F')?&gt;</code></td>
</tr>
<tr>
<td>Months</td>
<td><code>&lt;?count(current-group())?&gt;</code></td>
</tr>
<tr>
<td>End TmpRng</td>
<td><code>&lt;?end for-each-group?&gt;</code></td>
</tr>
</tbody>
</table>

Note the following about the form field tags:

- The `<?for-each-group:temp;floor(degree div 10)?>` is the regrouping tag. It specifies that for the existing `<temp>` group, the elements are to be regrouped by the expression, `floor(degree div 10)`. The `floor` function is an XSL function that returns the highest integer that is not greater than the argument (for example, 1.2 returns 1, 0.8 returns 0).

  In this case, it returns the value of the `<degree>` element, which is then divided by 10. This will generate the following values from the XML data: 1, 1, 1, 2, 3, 3, 3, 2, 2, 1, and 0.
These are sorted, so that when processed, the following four groups will be created: 0, 1, 2, and 3.

- The `<xsl:concat floor(degree div 10)*10, 'F' to ', floor(degree div 10)*10+10, 'F'/>` displays the temperature ranges in the row header in increments of 10. The expression concatenates the value of the current group times 10 with the value of the current group times 10 plus 10.

Therefore, for the first group, 0, the row heading displays 0 to (0 +10), or "0 F to 10 F".

- The `<xsl:count current-group() ?>` uses the count function to count the members of the current group (the number of temperatures that satisfy the range).

- The `<xsl:end for-each-group ?>` tag closes out the grouping.

## Variables, Parameters, and Properties

### Using Variables

Updateable variables differ from standard XSL variables `<xsl:variable>` in that they are updateable during the template application to the XML data. This allows you to create many new features in your templates that require updateable variables.

The variables use a "set and get" approach for assigning, updating, and retrieving values.

Use the following syntax to declare/set a variable value:

```xml
<xsl:script-set-variable($XDOCTX, 'variable name', value)>
```

Use the following syntax to retrieve a variable value:

```xml
<xsl:script-get-variable($XDOCTX, 'variable name')>
```

You can use this method to perform calculations. For example:

```xml
<xsl:script-set-variable($XDOCTX, 'x', $XDOCTX, 'x' + 1)>
```

This sets the value of variable 'x' to its original value plus 1, much like using "x = x + 1".

The `$XDOCTX` specifies the global document context for the variables. In a multi-threaded environment there may be many transformations occurring at the same time, therefore the variable must be assigned to a single transformation.

See the section on Running Totals, page 2-70 for an example of the usage of updateable variables.

### Defining Parameters

You can pass runtime parameter values into your template. These can then be referenced throughout the template to support many functions. For example, you can filter data in the template, use a value in a conditional formatting block, or pass property values (such as security settings) into the final document.

**Note:** The Oracle Applications concurrent manager does not support passing parameter values into the template. The parameters must be passed programmatically using the APIs as described below.

### Using a parameter in a template

1. Declare the parameter in the template.
Use the following syntax to declare the parameter:

```xml
<xsl:param name="PARAMETERNAME" select="DEFAULT" xdofo:ctx="begin"/>
```

where

- `PARAMETERNAME` is the name of the parameter
- `DEFAULT` is the default value for the parameter (the select statement is optional)
- `xdofo:ctx="begin"` is a required string to push the parameter declaration to the top of the template at runtime so that it can be referred to globally in the template.

The syntax must be declared in the Help Text field of a form field. The form field can be placed anywhere in the template.

2. Refer to the parameter in the template by prefixing the name with a "$" character. For example, if you declare the parameter name to be "InvThresh", then reference the value using "$InvThresh".

3. At runtime, pass the parameter to the XML Publisher engine programmatically.

   Prior to calling either the FOPProcessor API (Core) or the TemplateHelper API (E-Business Suite) create a Properties class and assign a property to it for the parameter value as follows:

   ```java
   Properties prop = new Properties();
   prop.put("xslt.InvThresh", "1000");
   ``

   For more information, see Calling XML Publisher APIs, page 10-1.

**Example: Passing an invoice threshold parameter**

This example illustrates how to declare a parameter in your template that will filter your data based on the value of the parameter.

The following XML sample lists invoice data:

```xml
<INVOICES>
  <INVOICE>
    <INVOICE_NUM>981110</INVOICE_NUM>
    <AMOUNT>1100</AMOUNT>
  </INVOICE>
  <INVOICE>
    <INVOICE_NUM>981111</INVOICE_NUM>
    <AMOUNT>250</AMOUNT>
  </INVOICE>
  <INVOICE>
    <INVOICE_NUM>981112</INVOICE_NUM>
    <AMOUNT>8343</AMOUNT>
  </INVOICE>
  . . .
</INVOICES>
```

The following figure displays a template that accepts a parameter value to limit the invoices displayed in the final document based on the parameter value.

**InvThresh Declaration**

<table>
<thead>
<tr>
<th>Invoice Number</th>
<th>Invoice Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>FE IF 13222-2</td>
<td>$100.00</td>
</tr>
<tr>
<td>Field</td>
<td>Form Field Help Text Entry</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>InvThreshDeclaration</td>
<td><code>&lt;xsl:param name=&quot;InvThresh&quot; xdofo:ctx=&quot;begin&quot;/&gt;</code></td>
</tr>
<tr>
<td>FE</td>
<td><code>&lt;?for-each:INVOICE?&gt;</code></td>
</tr>
<tr>
<td>IF</td>
<td><code>&lt;?if:AMOUNT&gt;$InvThresh?&gt;</code></td>
</tr>
<tr>
<td>$100.00</td>
<td><code>&lt;?AMOUNT?&gt;</code></td>
</tr>
<tr>
<td>EI</td>
<td><code>&lt;?end if?&gt;</code></td>
</tr>
<tr>
<td>EFE</td>
<td><code>&lt;?end for-each?&gt;</code></td>
</tr>
</tbody>
</table>

In this template, only INVOICE elements with an AMOUNT greater than the InvThresh parameter value will be displayed. If we pass in a parameter value of 1,000, the following output shown in the following figure will result:

<table>
<thead>
<tr>
<th>Invoice Number</th>
<th>Invoice Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>981110</td>
<td>1100</td>
</tr>
<tr>
<td>981112</td>
<td>8343</td>
</tr>
</tbody>
</table>

Notice the second invoice does not display because its amount was less than the parameter value.

**Setting Properties**

XML Publisher properties that are available in the XML Publisher Configuration file can alternatively be embedded into the RTF template. The properties set in the template are resolved at runtime by the XML Publisher engine. You can either hard code the values in the template or embed the values in the incoming XML data. Embedding the properties in the template avoids the use of the configuration file.

*Note:* See XML Publisher Configuration File, page 8-2 for more information about the XML Publisher Configuration file and the available properties.

For example, if you use a nonstandard font in your template, rather than specify the font location in the configuration file, you can embed the font property inside the template. If you need to secure the generated PDF output, you can use the XML Publisher PDF security properties and obtain the password value from the incoming XML data.

To add an XML Publisher property to a template, use the Microsoft Word **Properties** dialog (available from the **File** menu), and enter the following information:

**Name** - enter the XML Publisher property name prefixed with "xdo-"
**Type** - select "Text"

**Value** - enter the property value. To reference an element from the incoming XML data, enter the path to the XML element enclosed by curly braces. For example: {/root/password}

The following figure shows the Properties dialog:

![xdoprop.ttf Properties](image)

**Embedding a Font Reference**

For this example, suppose you want to use a font in the template called "XMLPScript". This font is not available as a regular font on your server, therefore you must tell XML Publisher where to find the font at runtime. You tell XML Publisher where to find the font by setting the "font" property. Assume the font is located in "/tmp/fonts", then you would enter the following in the Properties dialog:

**Name**: `xdo-font.XMLPScript.normal.normal`

**Type**: Text

**Value**: `trueType./tmp/fonts/XMLPScript.ttf`

When the template is applied to the XML data on the server, XML Publisher will look for the font in the /tmp/fonts directory. Note that if the template is deployed in multiple locations, you must ensure that the path is valid for each location.

For more information about setting font properties, see Font Definitions, page A-4.
Securing a PDF Output

For this example, suppose you want to use a password from the XML data to secure the PDF output document. The XML data is as follows:

```xml
<PO>
  <security>true</security>
  <password>welcome</password>
  <PO_Details>
    ..
  </PO_Details>
</PO>
```

In the Properties dialog set two properties: `pdf-security` to set the security feature as enabled or not, and `pdf-open-password` to set the password. Enter the following in the Properties dialog:

**Name:** xdo-pdf-security  
**Type:** Text  
**Value:** \{/PO/security\}

**Name:** xdo-pdf-open-password  
**Type:** Text  
**Value:** \{/PO/password\}

Storing the password in the XML data is not recommended if the XML will persist in the system for any length of time. To avoid this potential security risk, you can use a template parameter value that is generated and passed into the template at runtime.

For example, you could set up the following parameters:

- PDFSec - to pass the value for the xdo-pdf-security property
- PDFPWD - to pass the value for the password

You would then enter the following in the Properties dialog:

**Name:** xdo-pdf-security  
**Type:** Text  
**Value:** \{$PDFSec\}

**Name:** xdo-pdf-open-password  
**Type:** Text  
**Value:** \{$PDFPWD\}

For more information about template parameters, see Defining Parameters in Your Template, page 2-79.

Advanced Report Layouts

Batch Reports

It is a common requirement to print a batch of documents, such as invoices or purchase orders in a single PDF file. Because these documents are intended for different customers, each document will require that the page numbering be reset and that page
totals are specific to the document. If the header and footer display fields from the data (such as customer name) these will have to be reset as well.

XML Publisher supports this requirement through the use of a context command. This command allows you to define elements of your report to a specific section. When the section changes, these elements are reset.

The following example demonstrates how to reset the header and footer and page numbering within an output file:

The following XML sample is a report that contains multiple invoices:

```xml
...<LIST_G_INVOICE>
   <G_INVOICE>
      <BILL_CUST_NAME>Vision, Inc. </BILL_CUST_NAME>
      <TRX_NUMBER>2345678</TRX_NUMBER>
      ...
   </G_INVOICE>
   <G_INVOICE>
      <BILL_CUST_NAME>Oracle, Inc. </BILL_CUST_NAME>
      <TRX_NUMBER>2345685</TRX_NUMBER>
      ...
   </G_INVOICE>
...<LIST_G_INVOICE>
```

Each `G_INVOICE` element contains an invoice for a potentially different customer. To instruct XML Publisher to start a new section for each occurrence of the `G_INVOICE` element, add the `@section` command to the opening `for-each` statement for the group, using the following syntax:

```xml
<?for-each@section:group_name?>
```

where `group_name` is the name of the element for which you want to begin a new section.

For example, the for-each grouping statement for this example will be as follows:

```xml
<?for-each@section:G_INVOICE?>
```

The closing `?end for-each?>` tag is not changed.

The following figure shows a sample template. Note that the `G_INVOICE` group for-each declaration is still within the body of the report, even though the headers will be reset by the command.
The following table shows the values of the form fields from the example:

<table>
<thead>
<tr>
<th>Default Text Entry</th>
<th>Form Field Help Text</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>for-each G_INVOICE</td>
<td>&lt;?for-each@section:G_INVOICE?&gt;</td>
<td>Begins the G_INVOICE group, and defines the element as a Section. For each occurrence of G_INVOICE, a new section will be started.</td>
</tr>
<tr>
<td>&lt;?TRX_NUMBER?&gt;</td>
<td>N/A</td>
<td>Microsoft Word does not support form fields in the header, therefore the placeholder syntax for the TRX_NUMBER element is placed directly in the template.</td>
</tr>
<tr>
<td>end G_INVOICE</td>
<td>&lt;?end for-each?&gt;</td>
<td>Closes the G_INVOICE group.</td>
</tr>
</tbody>
</table>

Now for each new occurrence of the G_INVOICE element, a new section will begin. The page numbers will restart, and if header or footer information is derived from the data, it will be reset as well.

Cross-Tab Support

The columns of a cross-tab report are data dependent. At design-time you do not know how many columns will be reported, or what the appropriate column headings will be. Moreover, if the columns should break onto a second page, you need to be able to define the row label columns to repeat onto subsequent pages. The following example shows how to design a simple cross-tab report that supports these features.

This example uses the following XML sample:
<ROWSET>
  <RESULTS>
    <INDUSTRY>Motor Vehicle Dealers</INDUSTRY>
    <YEAR>2005</YEAR>
    <QUARTER>Q1</QUARTER>
    <SALES>1000</SALES>
  </RESULTS>
  <RESULTS>
    <INDUSTRY>Motor Vehicle Dealers</INDUSTRY>
    <YEAR>2005</YEAR>
    <QUARTER>Q2</QUARTER>
    <SALES>2000</SALES>
  </RESULTS>
  <RESULTS>
    <INDUSTRY>Motor Vehicle Dealers</INDUSTRY>
    <YEAR>2004</YEAR>
    <QUARTER>Q1</QUARTER>
    <SALES>3000</SALES>
  </RESULTS>
  <RESULTS>
    <INDUSTRY>Motor Vehicle Dealers</INDUSTRY>
    <YEAR>2004</YEAR>
    <QUARTER>Q2</QUARTER>
    <SALES>3000</SALES>
  </RESULTS>
  <RESULTS>
    <INDUSTRY>Motor Vehicle Dealers</INDUSTRY>
    <YEAR>2003</YEAR>
    ...
  </RESULTS>
  <RESULTS>
    <INDUSTRY>Home Furnishings</INDUSTRY>
    ...
  </RESULTS>
  <RESULTS>
    <INDUSTRY>Electronics</INDUSTRY>
    ...
  </RESULTS>
  <RESULTS>
    <INDUSTRY>Food and Beverage</INDUSTRY>
    ...
  </RESULTS>
</ROWSET>

From this XML we will generate a report that shows each industry and totals the sales by year as shown in the following figure:

<table>
<thead>
<tr>
<th>Industry</th>
<th>2005</th>
<th>2004</th>
<th>2003</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motor Vehicle Dealers</td>
<td>3000</td>
<td>6000</td>
<td>1200</td>
</tr>
<tr>
<td>Home Furnishings</td>
<td>3200</td>
<td>7770</td>
<td>3300</td>
</tr>
<tr>
<td>Electronics</td>
<td>9000</td>
<td>9000</td>
<td>4300</td>
</tr>
<tr>
<td>Food and Beverage</td>
<td>1200</td>
<td>900</td>
<td>5600</td>
</tr>
</tbody>
</table>
The template to generate this report is shown in the following figure. The form field entries are shown in the subsequent table.

The form fields in the template have the following values:

<table>
<thead>
<tr>
<th>Default Text Entry</th>
<th>Form Field Help Text</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>header column</td>
<td>&lt;?horizontal-break-table:1?&gt;</td>
<td>Defines the first column as a header that should repeat if the table breaks across pages. For more information about this syntax, see Defining Columns to Repeat Across Pages, page 2-88.</td>
</tr>
<tr>
<td>for:</td>
<td>&lt;?for-each-group@column: RESULTS;YEAR?&gt;</td>
<td>Uses the regrouping syntax (see Regrouping the XML Data, page 2-72) to group the data by YEAR; and the @column context command to create a table column for each group (YEAR). For more information about context commands, see Using the Context Commands, page 2-107.</td>
</tr>
<tr>
<td>YEAR</td>
<td>&lt;?YEAR?&gt;</td>
<td>Placeholder for the YEAR element.</td>
</tr>
<tr>
<td>end</td>
<td>&lt;?end for-each-group?&gt;</td>
<td>Closes the for-each-group loop.</td>
</tr>
<tr>
<td>for:</td>
<td>&lt;?for-each-group:RESULTS;INDUSTRY?&gt;</td>
<td>Begins the group to create a table row for each INDUSTRY.</td>
</tr>
<tr>
<td>INDUSTRY</td>
<td>&lt;?INDUSTRY?&gt;</td>
<td>Placeholder for the INDUSTRY element.</td>
</tr>
<tr>
<td>for:</td>
<td>&lt;?for-each-group@cell:current-group();YEAR?&gt;</td>
<td>Uses the regrouping syntax (see Regrouping the XML Data, page 2-72) to group the data by YEAR; and the @cell context command to create a table cell for each group (YEAR).</td>
</tr>
<tr>
<td>sum(Sales)</td>
<td>&lt;?sum(current-group())/SALES)</td>
<td>Sums the sales for the current group (YEAR).</td>
</tr>
<tr>
<td>end</td>
<td>&lt;?end for-each-group?&gt;</td>
<td>Closes the for-each-group statement.</td>
</tr>
<tr>
<td>end</td>
<td>&lt;?end for-each-group?&gt;</td>
<td>Closes the for-each-group statement.</td>
</tr>
</tbody>
</table>

Note that only the first row uses the @column context to determine the number of columns for the table. All remaining rows need to use the @cell context to create the table cells for the column. (For more information about context commands, see Using the Context Commands, page 2-107.)
Dynamic Data Columns

The ability to construct dynamic data columns is a very powerful feature of the RTF template. Using this feature you can design a template that will correctly render a table when the number of columns required by the data is variable.

For example, you are designing a template to display columns of test scores within specific ranges. However, you do not how many ranges will have data to report. You can define a dynamic data column to split into the correct number of columns at runtime.

Use the following tags to accommodate the dynamic formatting required to render the data correctly:

- **Dynamic Column Header**
  
  ```xml
  <?split-column-header:group element name?>
  ```

  Use this tag to define which group to split for the column headers of a table.

- **Dynamic Column**
  
  ```xml
  <?split-column-data:group element name?>
  ```

  Use this tag to define which group to split for the column data of a table.

- **Dynamic Column Width**
  
  ```xml
  <?split-column-width:name?> or
  <?split-column-width:@width?>
  ```

  Use one of these tags to define the width of the column when the width is described in the XML data. The width can be described in two ways:

  - An XML element stores the value of the width. In this case, use the syntax
    ```xml
    <?split-column-width:name?>,
    ```
    where *name* is the XML element tag name that contains the value for the width.

  - If the element defined in the split-column-header tag contains a width attribute, use the syntax `<?split-column-width:@width?>` to use the value of that attribute.

- **Dynamic Column Width’s unit value (in points)**
  
  ```xml
  <?split-column-width-unit: value?>
  ```

  Use this tag to define a multiplier for the column width. If your column widths are defined in character cells, then you will need a multiplier value of \~6 to render the columns to the correct width in points. If the multiplier is not defined, the widths of the columns are calculated as a percentage of the total width of the table. This is illustrated in the following table:

<table>
<thead>
<tr>
<th>Width Definition</th>
<th>Column 1 (Width = 10)</th>
<th>Column 2 (Width = 12)</th>
<th>Column 3 (Width = 14)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplier not present - % width</td>
<td>10/10+12+14*100 28%</td>
<td>%Width = 33%</td>
<td>%Width = 39%</td>
</tr>
<tr>
<td>Multiplier = 6 - width 60 pts</td>
<td>72 pts</td>
<td>84 pts</td>
<td></td>
</tr>
</tbody>
</table>

Defining Columns to Repeat Across Pages

If your table columns expand horizontally across more than one page, you can define how many row heading columns you want to repeat on every page. Use the following syntax to specify the number of columns to repeat:
where *number* is the number of columns (starting from the left) to repeat.

Note that this functionality is supported for PDF output only.

### Example of Dynamic Data Columns

A template is required to display test score ranges for school exams. Logically, you want the report to be arranged as shown in the following table:

<table>
<thead>
<tr>
<th>Test Score</th>
<th>Test Score Range 1</th>
<th>Test Score Range 2</th>
<th>Test Score Range 3</th>
<th>...Test Score Range n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Category</td>
<td># students in Range 1</td>
<td># students in Range 2</td>
<td># students in Range 3</td>
<td># of students in Range n</td>
</tr>
</tbody>
</table>

but you do not know how many Test Score Ranges will be reported. The number of Test Score Range columns is dynamic, depending on the data.

The following XML data describes these test scores. The number of occurrences of the element `<TestScoreRange>` will determine how many columns are required. In this case there are five columns: 0-20, 21-40, 41-60, 61-80, and 81-100. For each column there is an amount element (`<NumOfStudents>`) and a column width attribute (`<TestScore width="15">`).

```xml
<?xml version="1.0" encoding="utf-8"?>
<TestScoreTable>
  <TestScores>
    <TestCategory>Mathematics</TestCategory>
    <TestScore width="15">
      <TestScoreRange>0-20</TestScoreRange>
      <NumOfStudents>30</NumOfStudents>
    </TestScore>
    <TestScore width="20">
      <TestScoreRange>21-40</TestScoreRange>
      <NumOfStudents>45</NumOfStudents>
    </TestScore>
    <TestScore width="15">
      <TestScoreRange>41-60</TestScoreRange>
      <NumOfStudents>50</NumOfStudents>
    </TestScore>
    <TestScore width="20">
      <TestScoreRange>61-80</TestScoreRange>
      <NumOfStudents>102</NumOfStudents>
    </TestScore>
    <TestScore width="15">
      <TestScoreRange>81-100</TestScoreRange>
      <NumOfStudents>22</NumOfStudents>
    </TestScore>
  </TestScores>
</TestScoreTable>
```

Using the dynamic column tags in form fields, set up the table in two columns as shown in the following figure. The first column, "Test Score" is static. The second column, "Column Header and Splitting" is the dynamic column. At runtime this column will split according to the data, and the header for each column will be appropriately populated. The Default Text entry and Form Field Help entry for each field are listed in

Creating an RTF Template 2-89
the table following the figure. (See Form Field Method, page 2-7 for more information on using form fields).

<table>
<thead>
<tr>
<th>Test Score</th>
<th>Column Header and Splitting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group:TestScores</td>
<td>Test Category</td>
</tr>
<tr>
<td></td>
<td>Content and Splitting end TestScores</td>
</tr>
</tbody>
</table>

**Default Text Entry**

| Group:TestScores | <for-each:TestScores?> |
| Test Category | <TestCategory?> |
| Column Header and Splitting | <split-column-header:TestScore?> <split-column-width:@width?><TestScoreRange>% |
| Content and Splitting | <split-column-data:TestScore?> <NumofStudents?> |
| end:TestScores | <end for-each?> |

- **Test Score** is the boilerplate column heading.
- Test Category is the placeholder for the `<TestCategory>` data element, that is, "Mathematics," which will also be the row heading.
- The second column is the one to be split dynamically. The width you specify will be divided by the number of columns of data. In this case, there are 5 data columns.
- The second column will contain the dynamic "range" data. The width of the column will be divided according to the split column width. Because this example does not contain the unit value tag (`<?split-column-width-unit:value?>`), the column will be split on a percentage basis. Wrapping of the data will occur if required.

  **Note:** If the tag (`<?split-column-width-unit:value?>`) were present, then the columns would have a specific width in points. If the total column widths were wider than the allotted space on the page, then the table would break onto another page.

  The "horizontal-break-table" tag could then be used to specify how many columns to repeat on the subsequent page. For example, a value of "1" would repeat the column "Test Score" on the subsequent page, with the continuation of the columns that did not fit on the first page.

The template will render the output shown in the following figure:

<table>
<thead>
<tr>
<th>Test Score</th>
<th>0-20</th>
<th>21-40</th>
<th>41-60</th>
<th>61-80</th>
<th>81-100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mathematics</td>
<td>30</td>
<td>45</td>
<td>50</td>
<td>102</td>
<td>22</td>
</tr>
</tbody>
</table>
Number and Date Formatting

Number Formatting

XML Publisher supports two methods for specifying the number format:

- Microsoft Word’s Native number format mask
- Oracle’s format-number function

Note: You can also use the native XSL format-number function to format numbers. See: Native XSL Number Formatting, page 2-111.

Use only one of these methods. If the number format mask is specified using both methods, the data will be formatted twice, causing unexpected behavior.

The group separator and the number separator will be set at runtime based on the template locale. This is applicable for both the Oracle format mask and the MS format mask.

Data Source Requirements

To use the Oracle format mask or the Microsoft format mask, the numbers in your data source must be in a raw format, with no formatting applied (for example: 1000.00). If the number has been formatted for European countries (for example: 1.000,00) the format will not work.

Note: The XML Publisher parser requires the Java BigDecimal string representation. This consists of an optional sign ("-") followed by a sequence of zero or more decimal digits (the integer), optionally followed by a fraction, and optionally followed by an exponent. For example: -123456.3455e-3.

Translation Considerations

If you are designing a template to be translatable, using currency in the Microsoft format mask is not recommended unless you want the data reported in the same currency for all translations. Using the MS format mask sets the currency in the template so that it cannot be updated at runtime.

Instead, use the Oracle format mask. For example, L999G999G999D99, where "L" will be replaced by the currency symbol based on the locale at runtime.

Do not include "%" in the format mask because this will fix the location of the percent sign in the number display, while the desired position could be at the beginning or the end of a number, depending on the locale.

Using the Microsoft Number Format Mask

To format numeric values, use Microsoft Word’s field formatting features available from the Text Form Field Options dialog box. The following graphic displays an example:
To apply a number format to a form field:

1. Open the **Form Field Options** dialog box for the placeholder field.
2. Set the **Type** to Number.
3. Select the appropriate **Number format** from the list of options.

**Supported Microsoft Format Mask Definitions**

The following table lists the supported Microsoft format mask definitions:
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Location</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Number</td>
<td>Digit. Each explicitly set 0 will appear, if no other number occupies the position.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Format mask: 00.0000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data: 1.234</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Display: 01.234</td>
</tr>
<tr>
<td>#</td>
<td>Number</td>
<td>Digit. When set to #, only the incoming data is displayed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Format mask: #.#.#.#</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data: 1.234</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Display: 1.234</td>
</tr>
<tr>
<td>.</td>
<td>Number</td>
<td>Determines the position of the decimal separator. The decimal separator symbol used will be determined at runtime based on template locale. For example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Format mask: #,#0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data: 1234.56</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Display for English locale: 1,234.56</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Display for German locale: 1.234,56</td>
</tr>
<tr>
<td>-</td>
<td>Number</td>
<td>Determines placement of minus sign for negative numbers.</td>
</tr>
<tr>
<td>,</td>
<td>Number</td>
<td>Determines the placement of the grouping separator. The grouping separator symbol used will be determined at runtime based on template locale. For example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Format mask: #,#0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data: 1234.56</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Display for English locale: 1,234.56</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Display for German locale: 1.234,56</td>
</tr>
<tr>
<td>E</td>
<td>Number</td>
<td>Separates mantissa and exponent in a scientific notation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.##E+0 plus sign always shown for positive numbers</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.##E-0 plus sign not shown for positive numbers</td>
</tr>
<tr>
<td>;</td>
<td>Subpattern boundary</td>
<td>Separates positive and negative subpatterns. See Note below.</td>
</tr>
<tr>
<td>%</td>
<td>Prefix or Suffix</td>
<td>Multiply by 100 and show as percentage</td>
</tr>
<tr>
<td>'</td>
<td>Prefix or Suffix</td>
<td>Used to quote special characters in a prefix or suffix.</td>
</tr>
</tbody>
</table>

**Note:** Subpattern boundary: A pattern contains a positive and negative subpattern, for example, "#,#0.00;(#,#0.00)". Each subpattern has a prefix, numeric part, and suffix. The negative subpattern is optional. If absent, the positive subpattern prefixed with the localized minus sign ("-" in most locales) is used as the negative subpattern. That is, "0.00"
alone is equivalent to "0.00;0.00". If there is an explicit negative subpattern, it serves only to specify the negative prefix and suffix. The number of digits, minimal digits, and other characteristics are all the same as the positive pattern. That means that "#,##0.0(#)" produces precisely the same behavior as "+#,##0.00(#,##0.0#)".

**Using the Oracle Format Mask**

To apply the Oracle format mask to a form field:

1. Open the Form Field Options dialog box for the placeholder field.
2. Set the Type to "Regular text".
3. In the Form Field Help Text field, enter the mask definition according to the following example:
   
   ```
   <?format-number:fieldname;'999G999D99'>
   ```

   where

   - `fieldname` is the XML tag name of the data element you are formatting and
   - `999G999D99` is the mask definition.

   The following graphic shows an example Form Field Help Text dialog entry for the data element "empno":

![Form Field Help Text](image)

The following table lists the supported Oracle number format mask symbols and their definitions:
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Digit. Each explicitly set 0 will appear, if no other number occupies the position. Example: Format mask: 00.0000 Data: 1.234 Display: 01.2340</td>
</tr>
<tr>
<td>9</td>
<td>Digit. Returns value with the specified number of digits with a leading space if positive or a leading minus if negative. Leading zeros are blank, except for a zero value, which returns a zero for the integer part of the fixed-point number. Example: Format mask: 99.9999 Data: 1.234 Display: 1.234</td>
</tr>
<tr>
<td>C</td>
<td>Returns the ISO currency symbol in the specified position.</td>
</tr>
<tr>
<td>D</td>
<td>Determines the placement of the decimal separator. The decimal separator symbol used will be determined at runtime based on template locale. For example: Format mask: 9G999D99 Data: 1234.56 Display for English locale: 1,234.56 Display for German locale: 1.234,56</td>
</tr>
<tr>
<td>EEEE</td>
<td>Returns a value in scientific notation.</td>
</tr>
<tr>
<td>G</td>
<td>Determines the placement of the grouping (thousands) separator. The grouping separator symbol used will be determined at runtime based on template locale. For example: Format mask: 9G999D99 Data: 1234.56 Display for English locale: 1,234.56 Display for German locale: 1.234,56</td>
</tr>
<tr>
<td>L</td>
<td>Returns the local currency symbol in the specified position.</td>
</tr>
<tr>
<td>MI</td>
<td>Displays negative value with a trailing &quot;,&quot;.</td>
</tr>
<tr>
<td>PR</td>
<td>Displays negative value enclosed by &lt;&gt;</td>
</tr>
<tr>
<td>PT</td>
<td>Displays negative value enclosed by ()</td>
</tr>
<tr>
<td>S (before number)</td>
<td>Displays positive value with a leading &quot;+&quot; and negative values with a leading &quot;,&quot;</td>
</tr>
<tr>
<td>S (after number)</td>
<td>Displays positive value with a trailing &quot;+&quot; and negative value with a trailing &quot;,&quot;</td>
</tr>
</tbody>
</table>

**Date Formatting**

XML Publisher supports three methods for specifying the date format:

- Specify an explicit date format mask using Microsoft Word’s native date format mask.
• Specify an explicit date format mask using Oracle’s format-date function.
• Specify an abstract date format mask using Oracle’s abstract date format masks. (Recommended for multilingual templates.)

Only one method should be used. If both the Oracle and MS format masks are specified, the data will be formatted twice causing unexpected behavior.

Data Source Requirements
To use the Microsoft format mask or the Oracle format mask, the date from the XML data source must be in canonical format. This format is:

\[ YYYY-MM-DDThh:mm:ss+HH:MM \]

where

• \[ YYYY \] is the year
• \[ MM \] is the month
• \[ DD \] is the day
• \[ T \] is the separator between the date and time component
• \[ hh \] is the hour in 24-hour format
• \[ mm \] is the minutes
• \[ ss \] is the seconds
• \[ +HH:MM \] is the time zone offset from Universal Time (UTC), or Greenwich Mean Time

An example of this construction is:

\[ 2005-01-01T09:30:10+00:00 \]

The data after the "T" is optional, therefore the following date: \[ 2005-01-01 \] can be formatted using either date formatting option. Note that if you do not include the time zone offset, the time will be formatted to the UTC time.

Translation Considerations
If you are designing a template to be translatable, explicitly setting a date format mask is not recommended. This is because the date format mask is part of the template, and all published reports based on this template will have the same date format regardless of locale.

For translatable templates, it is recommended that you use the Oracle abstract date format.

If it is necessary to explicitly specify a format mask, the Oracle format mask is recommended over the MS format mask to ensure future compatibility.

Using the Microsoft Date Format Mask
To apply a date format to a form field:

1. Open the Form Field Options dialog box for the placeholder field.
2. Set the Type to Date, Current Date, or Current Time.
3. Select the appropriate Date format from the list of options.
If you do not specify the mask in the **Date format** field, the abstract format mask "MEDIUM" will be used as default. See Oracle Abstract Format Masks, page 2-101 for the description.

The following figure shows the Text Form Field Options dialog box with a date format applied:

![Text Form Field Options dialog box](image)

The following table lists the supported Microsoft date format mask components:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>The day of the month. Single-digit days will not have a leading zero.</td>
</tr>
<tr>
<td>dd</td>
<td>The day of the month. Single-digit days will have a leading zero.</td>
</tr>
<tr>
<td>dddd</td>
<td>The abbreviated name of the day of the week, as defined in AbbreviatedDayNames.</td>
</tr>
<tr>
<td>ddddd</td>
<td>The full name of the day of the week, as defined in DayNames.</td>
</tr>
<tr>
<td>M</td>
<td>The numeric month. Single-digit months will not have a leading zero.</td>
</tr>
<tr>
<td>MM</td>
<td>The numeric month. Single-digit months will have a leading zero.</td>
</tr>
<tr>
<td>MMM</td>
<td>The abbreviated name of the month, as defined in AbbreviatedMonthNames.</td>
</tr>
<tr>
<td>MMMM</td>
<td>The full name of the month, as defined in MonthNames.</td>
</tr>
<tr>
<td>yy</td>
<td>The year without the century. If the year without the century is less than 10, the year is displayed with a leading zero.</td>
</tr>
<tr>
<td>yyyy</td>
<td>The year in four digits.</td>
</tr>
<tr>
<td>g8</td>
<td>The period or era. This pattern is ignored if the date to be formatted does not have an associated period or era string.</td>
</tr>
<tr>
<td>h</td>
<td>The hour in a 12-hour clock. Single-digit hours will not have a leading zero.</td>
</tr>
<tr>
<td>hh</td>
<td>The hour in a 12-hour clock. Single-digit hours will have a leading zero.</td>
</tr>
<tr>
<td>H</td>
<td>The hour in a 24-hour clock. Single-digit hours will not have a leading zero.</td>
</tr>
<tr>
<td>HH</td>
<td>The hour in a 24-hour clock. Single-digit hours will have a leading zero.</td>
</tr>
<tr>
<td>Symbol</td>
<td>Meaning</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>m</td>
<td>The minute. Single-digit minutes will not have a leading zero.</td>
</tr>
<tr>
<td>mm</td>
<td>The minute. Single-digit minutes will have a leading zero.</td>
</tr>
<tr>
<td>s</td>
<td>The second. Single-digit seconds will not have a leading zero.</td>
</tr>
<tr>
<td>ss</td>
<td>The second. Single-digit seconds will have a leading zero.</td>
</tr>
<tr>
<td>f</td>
<td>Displays seconds fractions represented in one digit.</td>
</tr>
<tr>
<td>ff</td>
<td>Displays seconds fractions represented in two digits.</td>
</tr>
<tr>
<td>fff</td>
<td>Displays seconds fractions represented in three digits.</td>
</tr>
<tr>
<td>ffff</td>
<td>Displays seconds fractions represented in four digits.</td>
</tr>
<tr>
<td>fffff</td>
<td>Displays seconds fractions represented in five digits.</td>
</tr>
<tr>
<td>fffffff</td>
<td>Displays seconds fractions represented in six digits.</td>
</tr>
<tr>
<td>ffffffff</td>
<td>Displays seconds fractions represented in seven digits.</td>
</tr>
<tr>
<td>tt</td>
<td>The AM/PM designator defined in AMDesignator or PMDesignator, if any.</td>
</tr>
<tr>
<td>z</td>
<td>Displays the time zone offset for the system's current time zone in whole hours only. (This element can be used for formatting only)</td>
</tr>
<tr>
<td>zz</td>
<td>Displays the time zone offset for the system's current time zone in whole hours only. (This element can be used for formatting only)</td>
</tr>
<tr>
<td>zzz</td>
<td>Displays the time zone offset for the system's current time zone in hours and minutes.</td>
</tr>
<tr>
<td>:</td>
<td>The default time separator defined in TimeSeparator.</td>
</tr>
<tr>
<td>/</td>
<td>The default date separator defined in DateSeparator.</td>
</tr>
<tr>
<td>‘</td>
<td>Quoted string. Displays the literal value of any string between two ‘ characters.</td>
</tr>
<tr>
<td>“</td>
<td>Quoted string. Displays the literal value of any string between two “ characters.</td>
</tr>
</tbody>
</table>

**Using the Oracle Format Mask**

To apply the Oracle format mask to a date field:

1. Open the **Form Field Options** dialog box for the placeholder field.
2. Set the **Type** to Regular Text.
3. Select the **Add Help Text...** button to open the **Form Field Help Text** dialog.
4. Insert the following syntax to specify the date format mask:

```xml
<?format-date:date_string; 'ABSTRACT_FORMAT_MASK'; 'TIMEZONE'?>
```

or

```xml
<?format-date-and-calendar:date_string; 'ABSTRACT_FORMAT_MASK'; 'CALENDAR_NAME'; 'TIMEZONE'?>
```

where time zone is optional. The detailed usage of format mask, calendar and time zone is described below.
If no format mask is specified, the abstract format mask "MEDIUM" will be used as default.

Example form field help text entry:

```xml
<?format-date:hiredate;'YYYY-MM-DD'?>
```

The following table lists the supported Oracle format mask components:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Punctuation and quoted text are reproduced in the result.</td>
</tr>
<tr>
<td>/</td>
<td>AD indicator with or without periods.</td>
</tr>
<tr>
<td>.</td>
<td>AM indicator with or without periods.</td>
</tr>
<tr>
<td>:</td>
<td>BC indicator with or without periods.</td>
</tr>
<tr>
<td>&quot;text&quot;</td>
<td>Century. For example, 2002 returns 21; 2000 returns 20.</td>
</tr>
<tr>
<td>DAY</td>
<td>Name of day, padded with blanks to length of 9 characters.</td>
</tr>
<tr>
<td>D</td>
<td>Day of week (1-7).</td>
</tr>
<tr>
<td>DD</td>
<td>Day of month (1-31).</td>
</tr>
<tr>
<td>DDD</td>
<td>Day of year (1-366).</td>
</tr>
<tr>
<td>DL</td>
<td>Returns a value in the long date format.</td>
</tr>
<tr>
<td>DS</td>
<td>Returns a value in the short date format.</td>
</tr>
<tr>
<td>DY</td>
<td>Abbreviated name of day.</td>
</tr>
<tr>
<td>E</td>
<td>Abbreviated era name.</td>
</tr>
<tr>
<td>EE</td>
<td>Full era name.</td>
</tr>
<tr>
<td>FF[1..9]</td>
<td>Fractional seconds. Use the numbers 1 to 9 after FF to specify the number of digits in the fractional second portion of the datetime value returned.</td>
</tr>
<tr>
<td>HH</td>
<td>Hour of day (1-12).</td>
</tr>
<tr>
<td>HH12</td>
<td>Hour of day (1-12).</td>
</tr>
<tr>
<td>HH24</td>
<td>Hour of day (0-23).</td>
</tr>
<tr>
<td>Symbol</td>
<td>Meaning</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>MI</td>
<td>Minute (0-59).</td>
</tr>
<tr>
<td>MM</td>
<td>Month (01-12; JAN = 01).</td>
</tr>
<tr>
<td>MON</td>
<td>Abbreviated name of month.</td>
</tr>
<tr>
<td>MONTH</td>
<td>Name of month, padded with blanks to length of 9 characters.</td>
</tr>
<tr>
<td>PM</td>
<td>Meridian indicator with or without periods.</td>
</tr>
<tr>
<td>P.M.</td>
<td></td>
</tr>
<tr>
<td>RR</td>
<td>Lets you store 20th century dates in the 21st century using only two digits.</td>
</tr>
<tr>
<td>RRRR</td>
<td>Round year. Accepts either 4-digit or 2-digit input. If 2-digit, provides the same return as RR. If you don't want this functionality, then simply enter the 4-digit year.</td>
</tr>
<tr>
<td>SS</td>
<td>Seconds (0-59).</td>
</tr>
<tr>
<td>TZD</td>
<td>Daylight savings information. The TZD value is an abbreviated time zone string with daylight savings information. It must correspond to the region specified in TZR. Example: PST (for Pacific Standard Time) PDT (for Pacific Daylight Time)</td>
</tr>
<tr>
<td>TZH</td>
<td>Time zone hour. (See TZM format element.)</td>
</tr>
<tr>
<td>TZM</td>
<td>Time zone minute. (See TZH format element.) Example: 'HH:MI:SS.FFTZH:TZM'</td>
</tr>
<tr>
<td>TZR</td>
<td>Time zone region information. The value must be one of the time zone regions supported in the database. Example: PST (Pacific Standard Time)</td>
</tr>
<tr>
<td>WW</td>
<td>Week of year (1-53) where week 1 starts on the first day of the year and continues to the seventh day of the year.</td>
</tr>
<tr>
<td>W</td>
<td>Week of month (1-5) where week 1 starts on the first day of the month and ends on the seventh.</td>
</tr>
<tr>
<td>X</td>
<td>Local radix character.</td>
</tr>
<tr>
<td>YYYY</td>
<td>4-digit year.</td>
</tr>
<tr>
<td>YY</td>
<td>Last 2, or 1 digit(s) of year.</td>
</tr>
<tr>
<td>Y</td>
<td></td>
</tr>
</tbody>
</table>

**Default Format Mask**

If you do not want to specify a format mask with either the MS method or the Oracle method, you can omit the mask definition and use the default format mask. The default format mask is the MEDIUM abstract format mask from Oracle. (See Oracle Abstract Format Masks, page 2-101 for the definition.)

To use the default option using the Microsoft method, set the **Type** to Date, but leave the **Date format** field blank in the **Text Form Field Options** dialog.

To use the default option using the Oracle method, do not supply a mask definition to the "format-date" function call, for example:
Oracle Abstract Format Masks

The abstract date format masks reflect the default implementations of date/time formatting in the I18N library. When you use one of these masks, the output generated will depend on the locale associated with the report.

Specify the abstract mask using the following syntax:

<?format-date:fieldname;'MASK'?>

where fieldname is the XML element tag and MASK is the Oracle abstract format mask name

For example:

<?format-date:hiredate;'SHORT'?>
<?format-date:hiredate;'LONG_TIME_TZ'?>

The following table lists the abstract format masks and the sample output that would be generated for US locale:

<table>
<thead>
<tr>
<th>Mask</th>
<th>Output for US Locale</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHORT</td>
<td>2/31/99</td>
</tr>
<tr>
<td>MEDIUM</td>
<td>Dec 31, 1999</td>
</tr>
<tr>
<td>LONG</td>
<td>Friday, December 31, 1999</td>
</tr>
<tr>
<td>SHORT_TIME</td>
<td>12/31/99 6:15 PM</td>
</tr>
<tr>
<td>MEDIUM_TIME</td>
<td>Dec 31, 1999 6:15 PM</td>
</tr>
<tr>
<td>LONG_TIME</td>
<td>Friday, December 31, 1999 6:15 PM</td>
</tr>
<tr>
<td>SHORT_TIME_TZ</td>
<td>12/31/99 6:15 PM GMT</td>
</tr>
<tr>
<td>MEDIUM_TIME_TZ</td>
<td>Dec 31, 1999 6:15 PM GMT</td>
</tr>
<tr>
<td>LONG_TIME_TZ</td>
<td>Friday, December 31, 1999 6:15 PM GMT</td>
</tr>
</tbody>
</table>

Currency Formatting

XML Publisher enables you to define specific currency format masks to apply to your published data at runtime.

To utilize currency formatting, you must:

1. Define your currency formats in XML Publisher's Administration interface. See Defining Currency Formats, page 8-16.
2. Assign the Currency Format Set as a configuration property at the desired level (site, data definition, or template). It is available from the FO Processing Properties, page 8-6 list.
3. Enter the format-currency command in your RTF template to apply the format to the field at runtime.

To use the format-currency command:

1. In the form field dialog of the field you want to format, enter the following syntax:

<?format-currency:FIELD_NAME;'currency-format-code'?>
For example, to format an element from your XML file named CURRENT_BALANCE according to a currency code USD, enter the following in the form field for the element:

```
<?format-currency:CURRENT_BALANCE;'USD'?>
```

The currency code must correspond to a currency format that is defined in the Currency Format Set to be used with this report. The Currency Format Set can be specified at the site level, data definition level, or template level. For more information, see Defining Currency Formats, page 8-16.

## Calendar and Time Zone Support

### Calendar Specification

The term "calendar" refers to the calendar date displayed in the published report. The following types are supported:

- GREGORIAN
- ARABIC_HIJRAH
- ENGLISH_HIJRAH
- JAPANESE_IMPERIAL
- THAI_BUDDHA
- ROC_OFFICIAL (Taiwan)

Use one of the following methods to set the calendar type:

- Call the `format-date-and-calendar` function and declare the calendar type.
  
  For example:
  ```
  <?format-date-and-calendar:hiredate;'LONG_TIME_TZ';'ROC_OFFICIAL';?>
  ```
  
  The following graphic shows the output generated using this definition with locale set to zh-TW and time zone set to Asia/Taipei:

  中華民國88年12月31日 星期五 下午 2:15 台北

- Set the calendar type using the profile option XDO: Calendar Type (XDOCALENDAR_TYPE).

  **Note:** The calendar type specified in the template will override the calendar type set in the profile option.

### Time Zone Specification

There are two ways to specify time zone information:

- Call the `format-date` or `format-date-and-calendar` function with the Oracle format.
- Set the user profile option Client Timezone (CLIENT_TIMEZONE_ID) in Oracle Applications.

If no time zone is specified, UTC is used.
In the template, the time zone must be specified as a Java time zone string, for example, America/Los Angeles. The following example shows the syntax to enter in the help text field of your template:

```xml
<?format-date:hiredate;'LONG_TIME_TZ';'Asia/Shanghai'?>
```

### Using External Fonts

XML Publisher enables you to use fonts in your output that are not normally available on the server. To set up a new font for your report output, use the font to design your template on your client machine, then make it available on the server, and configure XML Publisher to access the font at runtime.

1. Use the font in your template.
   1. Copy the font to your `<WINDOWS_HOME>/fonts` directory.
   2. Open Microsoft Word and build your template.
   3. Insert the font in your template: Select the text or form field and then select the desired font from the font dialog box (Format > Font) or font drop down list.

   The following graphic shows an example of the form field method and the text method:

   ![Font example]

2. Place the font on the server.
   Place the font in a directory accessible to the formatting engine at runtime.

3. Set the XML Publisher "font" property.
   You can set the font property either in the XML Publisher Configuration file or directly in the template.

   **To set the property in the configuration file:**

   Update the XML Publisher configuration file "fonts" section with the font name and its location on the server. For example, the new entry for a TrueType font is structured as follows:

   ```xml
   <true_type path="\user\fonts\MyFontName.ttf"/>
   ```

   See XML Publisher Configuration File, page 8-2 for more information.

   **To set the property in the template:**

Now you can run your report and XML Publisher will use the font in the output as designed. For PDF output, the advanced font handling features of XML Publisher embed the external font glyphs directly into the final document. The embedded font only contains the glyphs required for the document and not the complete font definition. Therefore the document is completely self-contained, eliminating the need to have external fonts installed on the printer.

For an example implementation, see the white paper, "Check Printing Using Oracle XML Publisher," MetaLink note 312353.1. This document describes how to set up the MICR font used in check printing.

**Advanced Design Options**

If you have more complex design requirements, XML Publisher supports the use of XSL and XSL:FO elements, and has also extended a set of SQL functions.

RTF templates offer extremely powerful layout options using XML Publisher’s syntax. However, because the underlying technology is based on open W3C standards, such as XSL and XPATH, you are not limited by the functionality described in this guide. You can fully utilize the layout and data manipulation features available in these technologies.

**XPath Overview**

XPath is an industry standard developed by the World Wide Web Consortium (W3C). It is the method used to navigate through an XML document. XPath is a set of syntax rules for addressing the individual pieces of an XML document. You may not know it, but you have already used XPath; RTF templates use XPath to navigate through the XML data at runtime.

This section contains a brief introduction to XPath principles. For more information, see the W3C Web site: http://www.w3.org/TR/xpath

XPath follows the Document Object Model (DOM), which interprets an XML document as a tree of nodes. A node can be one of seven types:

- root
- element
- attribute
- text
- namespace
- processing instruction
- comment

Many of these elements are shown in the following sample XML, which contains a catalog of CDs:
<?xml version="1.0" encoding="UTF-8"?>
<! -  My CD Listing  - >
<CATALOG>
  <CD cattype=Folk>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
  </CD>
  <CD cattype=Rock>
    <TITLE>Hide Your Heart</TITLE>
    <ARTIST>Bonnie Tyler</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <PRICE>9.90</PRICE>
    <YEAR>1988</YEAR>
  </CD>
</CATALOG>

The root node in this example is CATALOG. CD is an element, and it has an attribute cattype. The sample contains the comment My CD Listing. Text is contained within the XML document elements.

**Locating Data**

Locate information in an XML document using location-path expressions.

A node is the most common search element you will encounter. Nodes in the example CATALOG XML include CD, TITLE, and ARTIST. Use a path expression to locate nodes within an XML document. For example, the following path returns all CD elements:

```
//CATALOG/CD
```

where

the double slash (//) indicates that all elements in the XML document that match the search criteria are to be returned, regardless of the level within the document.

the slash (/) separates the child nodes. All elements matching the pattern will be returned.

To retrieve the individual TITLE elements, use the following command:

```
/CATALOG/CD/TITLE
```

This example will return the following XML:

```
<CATALOG>
  <CD cattype=Folk>
    <TITLE>Empire Burlesque</TITLE>
  </CD>
  <CD cattype=Rock>
    <TITLE>Hide Your Heart</TITLE>
  </CD>
</CATALOG>
```

Further limit your search by using square brackets. The brackets locate elements with certain child nodes or specified values. For example, the following expression locates all CDs recorded by Bob Dylan:

```
/CATALOG/CD[ARTIST="Bob Dylan"]
```

Or, if each CD element did not have an PRICE element, you could use the following expression to return only those CD elements that include a PRICE element:

```
/CATALOG/CD[PRICE]
Use the bracket notation to leverage the attribute value in your search. Use the @ symbol to indicate an attribute. For example, the following expression locates all Rock CDs (all CDs with the cattype attribute value Rock):

```
//CD[@cattype="Rock"]
```

This returns the following data from the sample XML document:

```
<CD cattype=Rock>
  <TITLE>Hide Your Heart</TITLE>
  <ARTIST>Bonnie Tylor</ARTIST>
  <COUNTRY>UK</COUNTRY>
  <PRICE>9.90</PRICE>
  <YEAR>1988</YEAR>
</CD>
```

You can also use brackets to specify the item number to retrieve. For example, the first CD element is read from the XML document using the following XPath expression:

```
/CATALOG/CD[1]
```

The sample returns the first CD element:

```
<CD cattype=Folk>
  <TITLE>Empire Burlesque</TITLE>
  <ARTIST>Bob Dylan</ARTIST>
  <COUNTRY>USA</COUNTRY>
  <PRICE>10.90</PRICE>
  <YEAR>1985</YEAR>
</CD>
```

XPath also supports wildcards to retrieve every element contained within the specified node. For example, to retrieve all the CDs from the sample XML, use the following expression:

```
/CATALOG/*
```

You can combine statements with Boolean operators for more complex searches. The following expression retrieves all Folk and Rock CDs, thus all the elements from the sample:

```
//CD[@cattype="Folk"]//CD[@cattype="Rock"]
```

The pipe (|) is equal to the logical OR operator. In addition, XPath recognizes the logical OR and AND, as well as the equality operators: <=, <, >, =, ==, and !=. For example, we can find all CDs released in 1985 or later using the following expression:

```
/CATALOG/CD[YEAR >=1985]
```

**Starting Reference**

The first character in an XPath expression determines the point at which it should start in the XML tree. Statements beginning with a forward slash (/) are considered absolute. No slash indicates a relative reference. An example of a relative reference is:

```
CD/*
```

This statement begins the search at the current reference point. That means if the example occurred within a group of statements the reference point left by the previous statement would be utilized.

A noted earlier, double forward slashes (//) retrieve every matching element regardless of location in the document.
Context and Parent

To select current and parent elements, XPath recognizes the dot notation commonly used to navigate directories. Use a single period (.) to select the current node and use double periods (..) to return the parent of the current node. For example, to retrieve all child nodes of the parent of the current node, use:

```
../*
```

Therefore, to access all CDs from the sample XML, use the following expression:

```
/CATALOG/CD/..
```

You could also access all the CD titles released in 1988 using the following:

```
/CATALOG/CD/TITLE[../YEAR=1988]
```

The .. is used to navigate up the tree of elements to find the YEAR element at the same level as the TITLE, where it is then tested for a match against "1988". You could also use // in this case, but if the element YEAR is used elsewhere in the XML document, you may get erroneous results.

XPath is an extremely powerful standard when combined with RTF templates allowing you to use conditional formatting and filtering in your template.

Namespace Support

If your XML data contains namespaces, you must declare them in the template prior to referencing the namespace in a placeholder. Declare the namespace in the template using either the basic RTF method or in a form field. Enter the following syntax:

```
<?namespace:namespace name= namespace url?>
```

For example:

```
```

Once declared, you can use the namespace in the placeholder markup, for example: `<fsg:ReportName>`

Using the Context Commands

The XML Publisher syntax is simplified XSL instructions. This syntax, along with any native XSL commands you may use in your template, is converted to XSL-FO when you upload the template to the Template Manager. The placement of these instructions within the converted stylesheet determines the behavior of your template.

XML Publisher’s RTF processor places these instructions within the XSL-FO stylesheet according to the most common context. However, sometimes you need to define the context of the instructions differently to create a specific behavior. To support this requirement, XML Publisher provides a set of context commands that allow you to define the context (or placement) of the processing instructions. For example, using context commands, you can:

- Specify an if statement in a table to refer to a cell, a row, a column or the whole table.
- Specify a for-each loop to repeat either the current data or the complete section (to create new headers and footers and restart the page numbering)
- Define a variable in the current loop or at the beginning of the document.

You can specify a context for both processing commands using the XML Publisher syntax and those using native XSL.
To specify a context for a processing command using the simplified XML Publisher syntax, simply add @context to the syntax instruction. For example:

- `<?for-each@section:INVOICE?>` specifies that the group INVOICE should begin a new section for each occurrence. By adding the section context, you can reset the header and footer and page numbering.

- `<?if@column:VAT?>` specifies that the if statement should apply to the VAT column only.

To specify a context for an XSL command, add the `xdofo:ctx="context"` attribute to your tags to specify the context for the insertion of the instructions. The value of the context determines where your code is placed.

For example:

```xml
<xsl:for-each xdofo:ctx="section" select ="INVOICE">
  <xsl:attribute xdofo:ctx="inblock" name="background-color">red</xsl:attribute>
</xsl:for-each>
```

XML Publisher supports the following context types:
<table>
<thead>
<tr>
<th>Context</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>section</td>
<td>The statement affects the whole section including the header and footer. For example, a for-each@context context command creates a new section for each occurrence - with restarted page numbering and header and footer. See Batch Reports, page 2-83 for an example of this usage.</td>
</tr>
<tr>
<td>column</td>
<td>The statement will affect the whole column of a table. This context is typically used to show and hide table columns depending on the data. See Column Formatting, page 2-59 for an example.</td>
</tr>
<tr>
<td>cell</td>
<td>The statement will affect the cell of a table. This is often used together with @column in cross-tab tables to create a dynamic number of columns. See Cross-Tab Support, page 2-85 for an example.</td>
</tr>
<tr>
<td>block</td>
<td>The statement will affect multiple complete fo:blocks (RTF paragraphs). This context is typically used for if and for-each statements. It can also be used to apply formatting to a paragraph or a table cell. See Cell Highlighting, page 2-62 for an example.</td>
</tr>
<tr>
<td>inline</td>
<td>The context will become the single statement inside an fo:inline block. This context is used for variables.</td>
</tr>
<tr>
<td>incontext</td>
<td>The statement is inserted immediately after the surrounding statement. This is the default for &lt;?sort?&gt; statements that need to follow the surrounding for-each as the first element.</td>
</tr>
<tr>
<td>inblock</td>
<td>The statement becomes a single statement inside an fo:block (RTF paragraph). This is typically not useful for control statements (such as if and for-each) but is useful for statements that generate text, such as call-template.</td>
</tr>
<tr>
<td>inlines</td>
<td>The statement will affect multiple complete inline sections. An inline section is text that uses the same formatting, such as a group of words rendered as bold. See If Statements in Boilerplate Text, page 2-56.</td>
</tr>
<tr>
<td>begin</td>
<td>The statement will be placed at the beginning of the XSL stylesheet. This is required for global variables. See Defining Parameters, page 2-79.</td>
</tr>
<tr>
<td>end</td>
<td>The statement will be placed at the end of the XSL stylesheet.</td>
</tr>
</tbody>
</table>

The following table shows the default context for the XML Publisher commands:
<table>
<thead>
<tr>
<th>Command</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply-template</td>
<td>inline</td>
</tr>
<tr>
<td>attribute</td>
<td>inline</td>
</tr>
<tr>
<td>call-template</td>
<td>inblock</td>
</tr>
<tr>
<td>choose</td>
<td>block</td>
</tr>
<tr>
<td>for-each</td>
<td>block</td>
</tr>
<tr>
<td>if</td>
<td>block</td>
</tr>
<tr>
<td>import</td>
<td>begin</td>
</tr>
<tr>
<td>param</td>
<td>begin</td>
</tr>
<tr>
<td>sort</td>
<td>incontext</td>
</tr>
<tr>
<td>template</td>
<td>end</td>
</tr>
<tr>
<td>value-of</td>
<td>inline</td>
</tr>
<tr>
<td>variable</td>
<td>end</td>
</tr>
</tbody>
</table>

### Using XSL Elements

You can use any XSL element in your template by inserting the XSL syntax into a form field.

If you are using the basic RTF method, you cannot insert XSL syntax directly into your template. XML Publisher has extended the following XSL elements for use in RTF templates.

To use these in a basic-method RTF template, you must use the XML Publisher Tag form of the XSL element. If you are using form fields, use either option.

#### Apply a Template Rule

Use this element to apply a template rule to the current element's child nodes.

**XSL Syntax:** `<xsl:apply-templates select="name"/>`

**XML Publisher Tag:** `<?apply:name?>`

This function applies to `<xsl:template-match="n"> where $n$ is the element name.

#### Copy the Current Node

Use this element to create a copy of the current node.

**XSL Syntax:** `<xsl:copy-of select="name"/>`

**XML Publisher Tag:** `<?copy-of:name?>`
Call Template

Use this element to call a named template to be inserted into or applied to the current template. For example, use this feature to render a table multiple times.

XSL Syntax: <xsl:call-template name="name" />
XML Publisher Tag: <?call-template:name?>

Template Declaration

Use this element to apply a set of rules when a specified node is matched.

XSL Syntax: <xsl:template name="name" />
XML Publisher Tag: <?template:name ?>

Variable Declaration

Use this element to declare a local or global variable.

XSL Syntax: <xsl:variable name="name" />
XML Publisher Tag: <?variable:name ?>

Example:

<xsl:variable name="color" select="red"/>
Assigns the value "red" to the "color" variable. The variable can then be referenced in the template.

Import Stylesheet

Use this element to import the contents of one style sheet into another.

Note: An imported style sheet has lower precedence than the importing style sheet.

XSL Syntax: <xsl:import href="url" />
XML Publisher Tag: <?import:url ?>

Define the Root Element of the Stylesheet

This and the <xsl:stylesheet> element are completely synonymous elements. Both are used to define the root element of the style sheet.

Note: An included style sheet has the same precedence as the including style sheet.

XSL Syntax: <xsl:stylesheet xmlns:x="url" />
XML Publisher Tag: <?namespace:x=url ?>

Note: The namespace must be declared in the template. See Namespace Support, page 2-107.

Native XSL Number Formatting

The native XSL format-number function takes the basic format:

format-number(number, format, [decimalformat])
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Required. Specifies the number to be formatted.</td>
</tr>
</tbody>
</table>
| format    | Required. Specifies the format pattern. Use the following characters to specify the pattern:  
  • # (Denotes a digit. Example: ####)  
  • 0 (Denotes leading and following zeros. Example: 0000.00)  
  • . (The position of the decimal point Example: ###.##)  
  • , (The group separator for thousands. Example: ###,###.##)  
  • % (Displays the number as a percentage. Example: ##%)  
  • ; (Pattern separator. The first pattern will be used for positive numbers and the second for negative numbers) |
| decimalformat | Optional. For more information on the decimal format please consult any basic XSLT manual. |

**Using FO Elements**

You can use the native FO syntax inside the Microsoft Word form fields.

For more information on XSL-FO see the W3C Website at http://www.w3.org/2002/08/XSLFOSummary.html

**Best Practices**

**Using Tables**

To optimize the exact placement of elements when the template is transformed into XSL, it is recommended that you use tables to define the placement and alignment.

Note the use of tables in the Payables Invoice Register:
A table is used in the header to place the image, the title, and the date in exact positions. By using a table, each element can be aligned within its own cell; thereby allowing a left alignment for the image, a center alignment for the title, and a right alignment for the date and page number.

A table is also used for the totals line of the report to achieve alignment with the entries in the Invoices table.

Tables used for formatting only can be hidden at runtime by turning off (hiding) the table gridlines.

### Using Subtemplates

It is likely that you will design several templates that contain common components or functionality, such as address formatting or string formatting functions. Rather than replicate this functionality in every template, you can place it in a subtemplate that is stored in the template manager and called from all the others.

If your template requires complex calculations or conditional formatting of tabular data, it is recommended that you place these instructions in a subtemplate to keep the primary template simple for your users to understand.

Using a subtemplate consists of four steps:

1. Create the subtemplate
2. Register the subtemplate in the Template Manager
3. Import the subtemplate to the main template
4. Call the subtemplate in your main template

### Create the Subtemplate

To define a template as a subtemplate, add the following tag to the beginning of the section to be included:

```
<?template:template_name?>
```
where

template_name is whatever name you choose for the section.

Note that in a single subtemplate file, you can have multiple `<?template:?>` entries, marking different segments you would like to include in other files.

**Register the Template in the Template Manager**

Register the subtemplate in the Template Manager. To define it as a subtemplate, select Subtemplate = YES.

For more information about the Template Manager, see Creating the Template, page 5-4.

**Import the Subtemplate to the Main Template**

Call a template from within another template by using the import command as follows:

`<?import:xdo://APPCODE.TEMPLATE_CODE.LANG.TERR?>`

where

APPCODE is the Application code for the template defined in the Template Manager.

TEMPLATE_CODE is the template code defined in the Template Manager.

LANG is the template language specified in the Template Manager.

TERR is the template territory specified in the Template Manager.

**Call the Subtemplate in Your Main Template**

Once you have imported the template, use the call template command to render its contents in the position specified, as follows:

`<?call template:template_name?>`

where

template_name is the name assigned in the template declaration of the subtemplate.

**Example**

In this example, your company address is a fixed string that appears in all your templates. Rather than reproduce the string in all your templates, you can place it in one template and reference it from all the others.

The common components template will contain the following:

```xml
<?template:MyAddress?>
R&G Corporation
500 Marine Parkway
Redwood Shores, CA 94065
<?end template?>
```

This defines the string in a function (or XSL template) called MyAddress. Save this template as RTF and add it to the Template Manager using the following criteria:

- Application Code: CUSTOM
- Template Code: COMMONCOMPONENTS
- Language: English
- Territory: United States of America
- Select "Yes" for Subtemplate

Now this subtemplate and any functions therein can be referenced from any other template.
To use this subtemplate in another template, first import it by inserting the following syntax at the top of the calling template:

```<?import:xdn://CUSTOM.COMMONCOMPONENTS.en.US?>```

At the position in the template where you want to display the address, enter:

```<?call template:MyAddress?>```

At runtime the string will be fetched from the subtemplate and rendered in the layout of the calling template.

This functionality is not limited to just strings, you can insert any valid RTF template functionality in a subtemplate, and even pass parameters from one to the other.
Creating a PDF Template

This chapter covers the following topics:

• PDF Template Overview
• Designing the Layout
• Adding Markup to the Template Layout
• Adding Page Numbers and Page Breaks
• Performing Calculations
• Completed PDF Template
• Runtime Behavior
• Creating a Template from a Downloaded PDF

PDF Template Overview

To create a PDF template, take any existing PDF document and apply the XML Publisher markup. Because the source of the PDF document does not matter, you have multiple design options. For example:

• Design the layout of your template using any application that generates documents that can be converted to PDF
• Scan a paper document to use as a template
• Download a PDF document from a third-party Web site

**Note:** The steps required to create a template from a third-party PDF depend on whether form fields have been added to the document. For more information, see Creating a Template from a Downloaded PDF, page 3-15.

If you are designing the layout, note that once you have converted to PDF, your layout is treated like a set background. When you mark up the template, you draw fields on top of this background. To edit the layout, you must edit your original document and then convert back to PDF.

For this reason, the PDF template is not recommended for documents that will require frequent updates to the layout. However, it is appropriate for forms that will have a fixed layout, such as invoices or purchase orders.
**Supported Modes**

XML Publisher supports Adobe Acrobat 5.0 (PDF specification version 1.4). If you are using Adobe Acrobat Professional 6.0 (or later), use the **Reduce File Size Option** (from the **File** menu) to save your file as Adobe Acrobat 5.0 compatible.

For PDF conversion, XML Publisher supports any PDF conversion utility, such as Adobe Acrobat Distiller.

**Designing the Layout**

To design the layout of your template you can use any desktop application that generates documents that can be converted to PDF. Or, scan in an original paper document to use as the background for the template.

The following is the layout for a sample purchase order. It was designed using Microsoft Word and converted to PDF using Adobe Acrobat Distiller.
The following is the XML data that will be used as input to this template:
Adding Markup to the Template Layout

After you have converted your document to PDF, you define form fields that will display the data from the XML input file. These form fields are placeholders for the data.

The process of associating the XML data to the PDF template is the same as the process for the RTF template. See: Associating the XML Data to the Template Layout: Associating the XML data to the template layout, page 2-3.

When you draw the form fields in Adobe Acrobat, you are drawing them on top of the layout that you designed. There is not a relationship between the design elements on
your template and the form fields. You therefore must place the fields exactly where you want the data to display on the template.

Creating a Placeholder

You can define a placeholder as text, a check box, or a radio button, depending on how you want the data presented.

Note: If you are using Adobe Acrobat 5.0, the Form Tool is available from the standard toolbar. If you are using Adobe Acrobat 6.0 or later, display the Forms Toolbar from the Tools menu by selecting Tools > Advanced Editing > Forms > Show Forms Toolbar.

Naming the Placeholder

When you enter a name for the placeholder, enter either the XML source field name or assign a different, unique name.

Note: The placeholder name must not contain the "." character.

If you assign a different name, you must map the template field to the data source field when you register the template in the Template Manager. Mapping requires that you load the XML schema. If you give the template field the same name as the XML source field, no mapping is required.

For information on mapping fields in the Template Manager, see Mapping PDF Template Fields, page 5-8.

Creating a Text Placeholder

To create a text placeholder in your PDF document:

Acrobat 5.0 Users:

1. Select the Form Tool from the Acrobat toolbar.
2. Draw a form field box in the position on the template where you want the field to display. Drawing the field opens the Field Properties dialog box.
3. In the Name field of the Field Properties dialog box, enter a name for the field.
4. Select Text from the Type drop down menu.
   
   You can use the Field Properties dialog box to set other attributes for the placeholder. For example, enforce maximum character size, set field data type, data type validation, visibility, and formatting.
5. If the field is not placed exactly where desired, drag the field for exact placement.

Acrobat 6.0 (and later) Users:

1. Select the Text Field Tool from the Forms Toolbar.
2. Draw a form field box in the position on the template where you want the field to display. Drawing the field opens the Text Field Properties dialog box.
3. On the General tab, enter a name for the placeholder in the Name field.
You can use the **Text Field Properties** dialog box to set other attributes for the placeholder. For example, enforce maximum character size, set field data type, data type validation, visibility, and formatting.

4. If the field is not placed exactly where desired, drag the field for exact placement.

**Supported Field Properties Options**

XML Publisher supports the following options available from the **Field Properties** dialog box. For more information about these options, see the Adobe Acrobat documentation.

- **Appearance**
  - Border Settings: color, background, width, and style
  - Text Settings: color, font, size
  - Common Properties: read only, required, visible/hidden, orientation (in degrees)
    
    (In Acrobat 6.0, these are available from the General tab)
  - Border Style

- **Options** tab
  - Multi-line
  - Scrolling Text

- **Format** tab - Number category options only
- **Calculate** tab - all calculation functions

**Creating a Check Box**

A check box is used to present options from which more than one can be selected. Each check box represents a different data element. You define the value that will cause the check box to display as "checked."

For example, a form contains a check box listing of automobile options such as Power Steering, Power Windows, Sunroof, and Alloy Wheels. Each of these represents a different element from the XML file. If the XML file contains a value of "Y" for any of these fields, you want the check box to display as checked. All or none of these options may be selected.

To create a check box field:

**Acrobat 5.0 Users:**

1. Draw the form field.
2. In the **Field Properties** dialog box, enter a **Name** for the field.
3. Select **Check Box** from the **Type** drop down list.
4. Select the **Options** tab.
5. In the **Export Value** field enter the value that the XML data field should match to enable the "checked" state.

   For the example, enter "Y" for each check box field.

**Acrobat 6.0 (and later) Users:**

1. Select the **Check Box Tool** from the Forms Toolbar.
2. Draw the check box field in the desired position.
3. On the **General** tab of the **Check Box Properties** dialog box, enter a **Name** for the field.
4. Select the **Options** tab.
5. In the **Export Value** field enter the value that the XML data field should match to enable the "checked" state.
   
   For the example, enter "Y" for each check box field.

**Creating a Radio Button Group**

A radio button group is used to display options from which only one can be selected.

For example, your XML data file contains a field called `<SHIPMENT_METHOD>`. The possible values for this field are "Standard" or "Overnight". You represent this field in your form with two radio buttons, one labeled "Standard" and one labeled "Overnight". Define both radio button fields as placeholders for the `<SHIPMENT_METHOD>` data field. For one field, define the "on" state when the value is "Standard". For the other, define the "on" state when the value is "Overnight".

To create a radio button group:

**Acrobat 5.0 Users:**

1. Draw the form field.
2. On the **Field Properties** dialog box, enter a **Name** for the field. Each radio button you define to represent this value can be named differently, but must be mapped to the same XML data field.
3. Select **Radio Button** from the **Type** drop down list.
4. Select the **Options** tab.
5. In the **Export Value** field enter the value that the XML data field should match to enable the "on" state.
   
   For the example, enter "Standard" for the field labeled "Standard". Enter "Overnight" for the field labeled "Overnight".

**Acrobat 6.0 (and later) Users:**

1. Select the **Radio Button Tool** from the Forms Toolbar.
2. Draw the form field in the position desired on the template.
3. On the **General** tab of the Radio Button Properties dialog, enter a **Name** for the field. Each radio button you define to represent this value can be named differently, but must be mapped to the same XML data field.
4. Select the **Options** tab.
5. In the **Export Value** field enter the value that the XML data field should match to enable the "on" state.
   
   For the example, enter "Standard" for the field labeled "Standard". Enter "Overnight" for the field labeled "Overnight".
Defining Groups of Repeating Fields

In the PDF template, you explicitly define the area on the page that will contain the repeating fields. For example, on the purchase order template, the repeating fields should display in the block of space between the Item header row and the Total field.

To define the area to contain the group of repeating fields:

1. Insert a form field at the beginning of the area that is to contain the group. (Acrobat 6.0 users select the Text Field Tool, then draw the form field.)
2. In the Name field of the Field Properties window, enter any unique name you choose. This field is not mapped.
3. Acrobat 5.0 users: Select Text from the Type drop down list.
4. In the Short Description field (Acrobat 5.0) or the Tooltip field (Acrobat 6.0) of the Field Properties window, enter the following syntax:
   
   ```xml
   <?rep_field="BODY_START"?>
   ```

5. Define the end of the group area by inserting a form field at the end of the area that is to contain the group.
6. In the Name field of the Field Properties window, enter any unique name you choose. This field is not mapped. Note that the name you assign to this field must be different from the name you assigned to the 'body start' field.
7. Acrobat 5.0 users: Select Text from the Type drop down list.
8. In the Short Description field (Acrobat 5.0) or the Tooltip field (Acrobat 6.0) of the Field Properties window, enter the following syntax:

   ```xml
   <?rep_field="BODY_END"?>
   ```

To define a group of repeating fields:

1. Insert a placeholder for the first element of the group.

   **Note:** The placement of this field in relationship to the BODY_START tag defines the distance between the repeating rows for each occurrence. See Placement of Repeating Fields, page 3-15.

2. For each element in the group, enter the following syntax in the Short Description field (Acrobat 5.0) or the Tooltip field (Acrobat 6.0):

   ```xml
   <?rep_field="T1_Gn"?>
   ```

   where n is the row number of the item on the template.

   For example, the group in the sample report is laid out in three rows.

   - For the fields belonging to the row that begins with "PO_LINE_NUM" enter
     ```xml
     <?rep_field="T1_G1"?>
     ```
   
   - For the fields belonging to the row that begins with "C_FLEX_ITEM_DISP" enter
     ```xml
     <?rep_field="T1_G2"?>
     ```
   
   - For the fields belonging to the row that begins with "C_SHIP_TO_ADDRESS" enter
     ```xml
     <?rep_field="T1_G3"?>
     ```
The following graphic shows the entries for the **Short Description/Tooltip** field:

3. (Optional) Align your fields. To ensure proper alignment of a row of fields, it is recommended that you use Adobe Acrobat’s alignment feature.

### Adding Page Numbers and Page Breaks

This section describes how to add the following page-features to your PDF template:

- Page Numbers
- Page Breaks

### Adding Page Numbers

To add page numbers, define a field in the template where you want the page number to appear and enter an initial value in that field as follows:

1. Decide the position on the template where you want the page number to be displayed.
2. Create a placeholder field called `@pagenum@` (see Creating a Text Placeholder, page 3-5).
3. Enter a starting value for the page number in the **Default** field. If the XML data includes a value for this field, the start value assigned in the template will be overridden. If no start value is assigned, it will default to 1.

The figure below shows the Field Properties dialog for a page number field:
Adding Page Breaks

You can define a page break in your template to occur after a repeatable field. To insert a page break after the occurrence of a specific field, add the following to the syntax in the Short Description field of the Field Properties dialog box (use the Tooltip field for Acrobat 6.0):

```
page_break="yes"
```

For example:

```
<?rep_field="T1_G3", page_break="yes"?>
```

The following example demonstrates inserting a page break in a template. The XML sample contains salaries of employees by department:

```
<?xml version="1.0"?>
<! - Generated by Oracle Reports version 6.0.8.22.0 - >
<ROOT>
  <LIST_G_DEPTNO>
    <G_DEPTNO>
      <DEPTNO>10</DEPTNO>
    </G_DEPTNO>
    <LIST_G_EMPNO>
      <G_EMPNO>
        <EMPNO>7782</EMPNO>
        <ENAME>CLARK</ENAME>
        <JOB>MANAGER</JOB>
        <SAL>2450</SAL>
      </G_EMPNO>
      <G_EMPNO>
```
We want to report the salary information for each employee by department as shown in the following template:
To insert a page break after each department, insert the page break syntax in the Short Description (or Tooltip field) for the SUMSALPERDEPTNO field as follows:

```xml
<?rep_field="T1_G3", page_break="yes"?>
```

The Field Properties dialog box for the field is shown in the following figure:

The sample report with data is shown in the following figure:
Performing Calculations
Adobe Acrobat provides a calculation function in the Field Properties dialog box. To create a field to display a calculated total on your report:

1. Create a text field to display the calculated total. Give the field any Name you choose.
2. In the Field Properties dialog box, select the Format tab.
3. Select Number from the Category list.
4. Select the Calculate tab.
5. Select the radio button next to "Value is the operation of the following fields:"
6. Select sum from the drop down list.
7. Select the Pick... button and select the fields that you want totaled.

**Completed PDF Template**

The following figure shows the completed PDF template:
Runtime Behavior

Placement of Repeating Fields

As already noted, the placement, spacing, and alignment of fields that you create on the template are independent of the underlying form layout. At runtime, XML Publisher places each repeating row of data according to calculations performed on the placement of the rows of fields that you created, as follows:

First occurrence:
The first row of repeating fields will display exactly where you have placed them on the template.

Second occurrence, single row:
To place the second occurrence of the group, XML Publisher calculates the distance between the BODY_START tag and the first field of the first occurrence. The first field of the second occurrence of the group will be placed this calculated distance below the first occurrence.

Second occurrence, multiple rows:
If the first group contains multiple rows, the second occurrence of the group will be placed the calculated distance below the last row of the first occurrence.
The distance between the rows within the group will be maintained as defined in the first occurrence.

Overflow Data

When multiple pages are required to accommodate the occurrences of repeating rows of data, each page will display identically except for the defined repeating area, which will display the continuation of the repeating data. For example, if the item rows of the purchase order extend past the area defined on the template, succeeding pages will display all data from the purchase order form with the continuation of the item rows.

Creating a Template from a Downloaded PDF

The steps for creating a template from a downloaded PDF are:

1. Register the Applications data source in the Template Manager.
2. Register the PDF form as a Template in the Template Manager.
3. Use the mapping feature to map the fields from the downloaded PDF form to your data source.

PDF forms downloaded from third party sources may or may not contain the form fields already defined. To determine if the form fields are defined, open the document in Adobe Acrobat and select the Form Tool (in Acrobat 6.0, select the Text Field Tool). If the form fields are defined, they will display in the document.

If the form fields are not defined, you must mark up the template. See Mark up the Layout, page 3-4 for instructions on inserting placeholders and defining groups of repeating fields.

If the form fields are defined, you are ready to upload the document to the Template Manager for field mapping.
This chapter covers the following topics:
- Introduction
- Structure of eText Templates
- Constructing the Data Tables
- Setup Command Tables
- Expressions, Control Structure, and Functions
- Identifiers, Operators, and Literals

Introduction

An eText template is an RTF-based template that is used to generate text output for Electronic Funds Transfer (EFT) and Electronic Data Interchange (EDI). At runtime, XML Publisher applies this template to an input XML data file to create an output text file that can be transmitted to a bank or other customer. Because the output is intended for electronic communication, the eText templates must follow very specific format instructions for exact placement of data.

Note: An EFT is an electronic transmission of financial data and payments to banks in a specific fixed-position format flat file (text).

EDI is similar to EFT except it is not only limited to the transmission of payment information to banks. It is often used as a method of exchanging business documents, such as purchase orders and invoices, between companies. EDI data is delimiter-based, and also transmitted as a flat file (text).

Files in these formats are transmitted as flat files, rather than printed on paper. The length of a record is often several hundred characters and therefore difficult to layout on standard size paper.

To accommodate the record length, the EFT and EDI templates are designed using tables. Each record is represented by a table. Each row in a table corresponds to a field in a record. The columns of the table specify the position, length, and value of the field.

These formats can also require special handling of the data from the input XML file. This special handling can be on a global level (for example, character replacement and sequencing) or on a record level (for example, sorting). Commands to perform...
these functions are declared in command rows. Global level commands are declared in setup tables.

At runtime, XML Publisher constructs the output file according to the setup commands and layout specifications in the tables.

**Prerequisites**

This section is intended for users who are familiar with EDI and EFT transactions audience for this section preparers of eText templates will require both functional and technical knowledge. That is, functional expertise to understand bank and country specific payment format requirements and sufficient technical expertise to understand XML data structure and eText specific coding syntax commands, functions, and operations.

**Structure of eText Templates**

There are two types of eText templates: fixed-position based (EFT templates) and delimiter-based (EDI templates). The templates are composed of a series of tables. The tables define layout and setup commands and data field definitions. The required data description columns for the two types of templates vary, but the commands and functions available are the same. A table can contain just commands, or it can contain commands and data fields.

The following graphic shows a sample from an EFT template to display the general structure of command and data rows:

---

4-2 Oracle XML Publisher User's Guide
Commands that apply globally, or commands that define program elements for the template, are "setup" commands. These must be specified in the initial table(s) of the template. Examples of setup commands are Template Type and Character Set.

In the data tables you provide the source XML data element name (or static data) and the specific placement and formatting definitions required by the receiving bank or entity. You can also define functions to be performed on the data and conditional statements.

The data tables must always start with a command row that defines the "Level." The Level associates the table to an element from the XML data file, and establishes the hierarchy. The data fields that are then defined in the table for the Level correspond to the child elements of the XML element.

The graphic below illustrates the relationship between the XML data hierarchy and the template Level. The XML element "RequestHeader" is defined as the Level. The data elements defined in the table ("FileID" and "Encryption") are children of the RequestHeader element.

The order of the tables in the template determines the print order of the records. At runtime the system loops through all the instances of the XML element corresponding to a table (Level) and prints the records belonging to the table. The system then moves on to the next table in the template. If tables are nested, the system will generate the nested records of the child tables before moving on to the next parent instance.

**Command Rows, Data Rows, and Data Column Header Rows**

The following figure shows the placement of Command Rows, Data Rows, and Data Column Header Rows:
Command rows are used to specify commands in the template. Command rows always have two columns: command name and command parameter. Command rows do not have column headings. The commands control the overall setup and record structures of the template.

Blank rows can be inserted anywhere in a table to improve readability. Most often they are used in the setup table, between commands. Blank rows are ignored by XML Publisher when the template is parsed.

**Data Column Header Rows**

Data column headers specify the column headings for the data fields (such as Position, Length, Format, Padding, and Comments). A column header row usually follows the Level command in a table (or the sorting command, if one is used). The column header row must come before any data rows in the table. Additional empty column header rows can be inserted at any position in a table to improve readability. The empty rows will be ignored at runtime.

The required data column header rows vary depending on the template type. See Structure of the Data Row, page 4-9.

**Data Rows**

Data rows contain the data fields to correspond to the column header rows.

The content of the data rows varies depending on the template type. See Structure of the Data Row, page 4-9.
**Constructing the Data Tables**

The data tables contain a combination of command rows and data field rows. Each data table must begin with a Level command row that specifies its XML element. Each record must begin with a New Record command that specifies the start of a new record, and the end of a previous record (if any).

The required columns for the data fields vary depending on the Template Type.

**Command Rows**

The command rows always have two columns: command name and command parameter. The supported commands are:

- Level
- New record
- Sort ascending
- Sort descending
- Display condition

The usage for each of these commands is described in the following sections.

**Level Command**

The level command associates a table with an XML element. The parameter for the level command is an XML element. The level will be printed once for each instance the XML element appears in the data input file.

The level commands define the hierarchy of the template. For example, Payment XML data extracts are hierarchical. A batch can have multiple child payments, and a payment can have multiple child invoices. This hierarchy is represented in XML as nested child elements within a parent element. By associating the tables with XML elements through the level command, the tables will also have the same hierarchical structure.

Similar to the closing tag of an XML element, the level command has a companion end-level command. The child tables must be defined between the level and end-level commands of the table defined for the parent element.

An XML element can be associated with only one level. All the records belonging to a level must reside in the table of that level or within a nested table belonging to that level. The end-level command will be specified at the end of the final table.

Following is a sample structure of an EFT file record layout:

- FileHeaderRecordA
  - BatchHeaderRecordA
  - BatchHeaderRecordB
    - PaymentRecordA
    - PaymentRecordB
    - InvoiceRecordA
  - Batch FooterRecordC
  - BatchFooterRecordD
• FileFooterRecordB

Following would be its table layout:

<LEVEL> RequestHeader
<NEW RECORD> FileHeaderRecordA
Data rows for the FileHeaderRecordA

<LEVEL> Batch
<NEW RECORD> BatchHeaderRecordA
Data rows for the BatchHeaderRecordA
<NEW RECORD> BatchHeaderRecordB
Data rows for the BatchHeaderRecordB

<LEVEL> Payment
<NEW RECORD> PaymentRecordA
Data rows for the PaymentRecordA
<NEW RECORD> PaymentRecordB
Data rows for the PaymentRecordB

<LEVEL> Invoice
<NEW RECORD> InvoiceRecordA
Data rows for the InvoiceRecordA
<END LEVEL> Invoice

<END LEVEL> Payment

<LEVEL> Batch
<NEW RECORD> BatchFooterRecordC
Data rows for the BatchFooterRecordC
<NEW RECORD> BatchFooterRecordD
Data rows for the BatchFooterRecordD
<END LEVEL> Batch
<LEVEL>
<NEW RECORD>
RequestHeader
FileFooterRecordB
Data rows for the FileFooterRecordB
<END LEVEL>
RequestHeader

Multiple records for the same level can exist in the same table. However, each table can only have one level defined. In the example above, the BatchHeaderRecordA and BatchHeaderRecordB are both defined in the same table. However, note that the END LEVEL for the Payment must be defined in its own separate table after the child element Invoice. The Payment END LEVEL cannot reside in the same table as the Invoice Level.

Note that you do not have to use all the levels from the data extract in your template. For example, if an extract contains the levels: RequestHeader > Batch > Payment > Invoice, you can use just the batch and invoice levels. However, the hierarchy of the levels must be maintained.

The table hierarchy determines the order that the records are printed. For each parent XML element, the records of the corresponding parent table are printed in the order they appear in the table. The system loops through the instances of the child XML elements corresponding to the child tables and prints the child records according to their specified order. The system then prints the records of the enclosing (end-level) parent table, if any.

For example, given the EFT template structure above, assume the input data file contains the following:

- Batch1
  - Payment1
    - Invoice1
    - Invoice2
  - Payment2
    - Invoice1
- Batch2
  - Payment1
    - Invoice1
    - Invoice2
    - Invoice3

This will generate the following printed records:
<table>
<thead>
<tr>
<th>Record Order</th>
<th>Record Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FileHeaderRecordA</td>
<td>One header record for the EFT file</td>
</tr>
<tr>
<td>2</td>
<td>BatchHeaderRecordA</td>
<td>For Batch1</td>
</tr>
<tr>
<td>3</td>
<td>BatchHeaderRecordB</td>
<td>For Batch1</td>
</tr>
<tr>
<td>4</td>
<td>PaymentRecordA</td>
<td>For Batch1, Payment1</td>
</tr>
<tr>
<td>5</td>
<td>PaymentRecordB</td>
<td>For Batch1, Payment1</td>
</tr>
<tr>
<td>6</td>
<td>InvoiceRecordA</td>
<td>For Batch1, Payment1, Invoice1</td>
</tr>
<tr>
<td>7</td>
<td>InvoiceRecordA</td>
<td>For Batch1, Payment1, Invoice2</td>
</tr>
<tr>
<td>8</td>
<td>PaymentRecordA</td>
<td>For Batch1, Payment2</td>
</tr>
<tr>
<td>9</td>
<td>PaymentRecordB</td>
<td>For Batch1, Payment2</td>
</tr>
<tr>
<td>10</td>
<td>InvoiceRecordA</td>
<td>For Batch1, Payment2, Invoice1</td>
</tr>
<tr>
<td>11</td>
<td>BatchFooterRecordC</td>
<td>For Batch1</td>
</tr>
<tr>
<td>12</td>
<td>BatchFooterRecordD</td>
<td>For Batch1</td>
</tr>
<tr>
<td>13</td>
<td>BatchHeaderRecordA</td>
<td>For Batch2</td>
</tr>
<tr>
<td>14</td>
<td>BatchHeaderRecordB</td>
<td>For Batch2</td>
</tr>
<tr>
<td>15</td>
<td>PaymentRecordA</td>
<td>For Batch2, Payment1</td>
</tr>
<tr>
<td>16</td>
<td>PaymentRecordB</td>
<td>For Batch2, Payment1</td>
</tr>
<tr>
<td>17</td>
<td>InvoiceRecordA</td>
<td>For Batch2, Payment1, Invoice1</td>
</tr>
<tr>
<td>18</td>
<td>InvoiceRecordA</td>
<td>For Batch2, Payment1, Invoice2</td>
</tr>
<tr>
<td>19</td>
<td>InvoiceRecordA</td>
<td>For Batch2, Payment1, Invoice3</td>
</tr>
<tr>
<td>20</td>
<td>BatchFooterRecordC</td>
<td>For Batch2</td>
</tr>
<tr>
<td>21</td>
<td>BatchFooterRecordD</td>
<td>For Batch2</td>
</tr>
<tr>
<td>22</td>
<td>FileFooterRecordB</td>
<td>One footer record for the EFT file</td>
</tr>
</tbody>
</table>

**New Record Command**

The new record command signifies the start of a record and the end of the previous one, if any. Every record in a template must start with the new record command. The record continues until the next new record command, or until the end of the table or the end of the level command.

A record is a construct for the organization of the elements belonging to a level. The record name is not associated with the XML input file.

A table can contain multiple records, and therefore multiple new record commands. All the records in a table are at the same hierarchy level. They will be printed in the order in which they are specified in the table.
The new record command can have a name as its parameter. This name becomes the name for the record. The record name is also referred to as the record type. The name can be used in the COUNT function for counting the generated instances of the record. See COUNT, page 4-21 function, for more information.

Consecutive new record commands (or empty records) are not allowed.

Sort Ascending and Sort Descending Commands

Use the sort ascending and sort descending commands to sort the instances of a level. Enter the elements you wish to sort by in a comma-separated list. This is an optional command. When used, it must come right after the (first) level command and it applies to all records of the level, even if the records are specified in multiple tables.

Display Condition Command

The display condition command specifies when the enclosed record or data field group should be displayed. The command parameter is a boolean expression. When it evaluates to true, the record or data field group is displayed. Otherwise the record or data field group is skipped.

The display condition command can be used with either a record or a group of data fields. When used with a record, the display condition command must follow the new record command. When used with a group of data fields, the display condition command must follow a data field row. In this case, the display condition will apply to the rest of the fields through the end of the record.

Consecutive display condition commands are merged as AND conditions. The merged display conditions apply to the same enclosed record or data field group.

Structure of the Data Rows

The output record data fields are represented in the template by table rows. In FIXED_POSITION_BASED templates, each row has the following attributes (or columns):

- Position
- Length
- Format
- Pad
- Data
- Comments

The first five columns are required and must appear in the order listed.

For DELIMITER_BASED templates, each data row has the following attributes (columns):

- Maximum Length
- Format
- Data
- Tag
- Comments
The first three columns are required and must be declared in the order stated.

In both template types, the Comments column is optional and ignored by the system. You can insert additional information columns if you wish, as all columns after the required ones are ignored.

The usage rules for these columns are as follows:

**Position**

Specifies the starting position of the field in the record. The unit is in number of characters. This column is only used with FIXED_POSITION_BASED templates.

**Length/Maximum Length**

Specifies the length of the field. The unit is in number of characters. For FIXED_POSITION_BASED templates, all the fields are fixed length. If the data is less than the specified length, it is padded. If the data is longer, it is truncated. The truncation always occurs on the right.

For DELIMITER_BASED templates, the maximum length of the field is specified. If the data exceeds the maximum length, it will be truncated. Data is not padded if it is less than the maximum length.

**Format**

Specifies the data type and format setting. There are three accepted data types: Alpha, Number, and Date. Refer to Field Level Key Words, page 4-24 for their usage.

Numeric data has two optional format settings: Integer and Decimal. Specify the optional settings with the Number data type as follows:

- Number, Integer
- Number, Decimal

The Integer format uses only the whole number portion of a numeric value and discards the decimal. The Decimal format uses only the decimal portion of the numeric value and discards the integer portion.

The Date data type format setting must always be explicitly stated. The format setting follows the SQL date styles, such as MMDDYY.

Some EDI (DELIMITER_BASED) formats use more descriptive data types. These are mapped to the three template data types in the following table:
<table>
<thead>
<tr>
<th>ASC X12 Data Type</th>
<th>Format Template Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>A - Alphabetic</td>
<td>Alpha</td>
</tr>
<tr>
<td>AN -Alphanumeric</td>
<td>Alpha</td>
</tr>
<tr>
<td>B - Binary</td>
<td>Number</td>
</tr>
<tr>
<td>CD - Composite data element</td>
<td>N/A</td>
</tr>
<tr>
<td>CH - Character</td>
<td>Alpha</td>
</tr>
<tr>
<td>DT - Date</td>
<td>Date</td>
</tr>
<tr>
<td>FS - Fixed-length string</td>
<td>Alpha</td>
</tr>
<tr>
<td>ID - Identifier</td>
<td>Alpha</td>
</tr>
<tr>
<td>IV - Incrementing Value</td>
<td>Number</td>
</tr>
<tr>
<td>Nn - Numeric</td>
<td>Number</td>
</tr>
<tr>
<td>PW - Password</td>
<td>Alpha</td>
</tr>
<tr>
<td>R - Decimal number</td>
<td>Number</td>
</tr>
<tr>
<td>TM - Time</td>
<td>Date</td>
</tr>
</tbody>
</table>

**Pad**

This applies to FIXED_POSITION_BASED templates only. Specify the padding side (L = left or R = right) and the character. Both numeric and alphanumeric fields can be padded. If this field is not specified, Numeric fields are left-padded with "0"; Alpha fields are right-padded with spaces.

Example usage:

- To pad a field on the left with a "0", enter the following in the Pad column field:
  
  \[ L, '0' \]

- To pad a field on the right with a space, enter the following the Pad column field:
  
  \[ R, ' ' \]

**Data**

Specifies the XML element from the data extract that is to populate the field. The data column can simply contain the XML tag name, or it can contain expressions and functions. For more information, see Expressions, Control Structure, and Functions, page 4-20.

**Tag**

Acts as a comment column for DELIMITER_BASED templates. It specifies the reference tag in EDIFACT formats, and the reference IDs in ASC X12.

**Comments**

Use this column to note any free form comments to the template. Usually this column is used to note the business requirement and usage of the data field.
Setup Command Tables

Setup Command Table

A template always begins with a table that specifies the setup commands. The setup commands define global attributes, such as template type and output character set and program elements, such as sequencing and concatenation.

The setup commands are:

- Template Type
- Output Character Set
- New Record Character
- Invalid Characters
- Replace Characters
- Define Level
- Define Sequence
- Define Concatenation

Some example setup tables are shown in the following figures:

XDO file name: XINT-01.rdf

<table>
<thead>
<tr>
<th></th>
<th>FIXING POSITION BASED</th>
<th>Mapping of Payment Format</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>International Payments EFT Format</td>
</tr>
</tbody>
</table>

**Format Setup:**

*Hint: Define formatting options...*

<table>
<thead>
<tr>
<th></th>
<th>FIXING POSITION BASED</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEMPLATE TYPE</td>
<td></td>
</tr>
<tr>
<td>OUTPUT CHARACTER SET</td>
<td>iso-8859-1</td>
</tr>
<tr>
<td>NEW RECORD CHARACTER</td>
<td>Carriage Return</td>
</tr>
</tbody>
</table>

|                |                               |
| INVALID CHARACTERS | ¥                        |
| REPLACE CHARACTERS | A0                        |
|                  | E0                        |
|                  | T0                        |
|                  | O0                        |
|                  | U0                        |

**Format Data Levels:**

*Hint: Define data levels that are needed in the format which do not exist in data extract...*

|                | PaymentsByPayDatePayee         |
| BASE LEVEL     | Payment                        |
| GROUPING CRITERIA | PaymentsDate, PayeeName        |
| END BASE LEVEL | PaymentsByPayDatePayee         |

|                | InvoicesByReportingCatAndAttrib |
| BASE LEVEL     | Invoice                        |
This command specifies the type of template. There are two types: FIXED_POSITION_BASED and DELIMITER_BASED.

Use the FIXED_POSITION_BASED templates for fixed-length record formats, such as EFTs. In these formats, all fields in a record are a fixed length. If data is shorter than the specified length, it will be padded. If longer, it will be truncated. The system specifies the default behavior for data padding and truncation. Examples of fixed position based formats are EFTs in Europe, and NACHA ACH file in the U.S.

In a DELIMITER_BASED template, data is never padded and only truncated when it has reached a maximum field length. Empty fields are allowed (when the data is null). Designated delimiters are used to separate the data fields. If a field is empty, two delimiters will appear next to each other. Examples of delimited-based templates are EDI formats such as ASC X12 820 and UN EDIFACT formats - PAYMUL, DIRDEB, and CREMUL.

In EDI formats, a record is sometimes referred to as a segment. An EDI segment is treated the same as a record. Start each segment with a new record command and give it a record name. You should have a data field specifying the segment name as part of the output data immediately following the new record command.
For DELIMITER_BASED templates, you insert the appropriate data field delimiters in separate rows between the data fields. After every data field row, you insert a delimiter row. You can insert a placeholder for an empty field by defining two consecutive delimiter rows.

Empty fields are often used for syntax reasons: you must insert placeholders for empty fields so that the fields that follow can be properly identified.

There are different delimiters to signify data fields, composite data fields, and end of record. Some formats allow you to choose the delimiter characters. In all cases you should use the same delimiter consistently for the same purpose to avoid syntax errors.

In DELIMITER_BASED templates, the <POSITION> and <PAD> columns do not apply. They are omitted from the data tables.

Some DELIMITER_BASED templates have minimum and maximum length specifications. In those cases Oracle Payments validates the length.

**Define Level Command**

Some formats require specific additional data levels that are not in the data extract. For example, some formats require that payments be grouped by payment date. Using the Define Level command, a payment date group can be defined and referenced as a level in the template, even though it is not in the input extract file.

When you use the Define Level command you declare a base level that exists in the extract. The Define Level command inserts a new level one level higher than the base level of the extract. The new level functions as a grouping of the instances of the base level.

The Define Level command is a setup command, therefore it must be defined in the setup table. It has three subcommands:

- **Base Level Command** - defines the level (XML element) from the extract that the new level is based on. The Define Level command must always have one and only one base level subcommand.

- **Grouping Criteria** - defines the XML extract elements that are used to group the instances of the base level to form the instances of the new level. The parameter of the grouping criteria command is a comma-separated list of elements that specify the grouping conditions.

  The order of the elements determines the hierarchy of the grouping. The instances of the base level are first divided into groups according to the values of the first criterion, then each of these groups is subdivided into groups according to the second criterion, and so on. Each of the final subgroups will be considered as an instance of the new level.

- **Group Sort Ascending or Group Sort Descending** - defines the sorting of the group. Insert the <GROUP SORT ASCENDING> or <GROUP SORT DESCENDING> command row anywhere between the <DEFINE LEVEL> and <END DEFINE LEVEL> commands. The parameter of the sort command is a comma-separated list of elements by which to sort the group.

For example, the following table shows five payments under a batch:
<table>
<thead>
<tr>
<th>Payment Instance</th>
<th>PaymentDate (grouping criterion 1)</th>
<th>PayeeName (grouping criterion 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Payment1</td>
<td>PaymentDate1</td>
<td>PayeeName1</td>
</tr>
<tr>
<td>Payment2</td>
<td>PaymentDate2</td>
<td>PayeeName1</td>
</tr>
<tr>
<td>Payment3</td>
<td>PaymentDate1</td>
<td>PayeeName2</td>
</tr>
<tr>
<td>Payment4</td>
<td>PaymentDate1</td>
<td>PayeeName1</td>
</tr>
<tr>
<td>Payment5</td>
<td>PaymentDate1</td>
<td>PayeeName3</td>
</tr>
</tbody>
</table>

In the template, construct the setup table as follows to create a level called "PaymentsByPayDatePayee" from the base level "Payment" grouped according to PaymentDate and Payee Name. Add the Group Sort Ascending command to sort ea:

```xml
<DEFINE LEVEL> PaymentsByPayDatePayee
<BASE LEVEL> Payment
<GROUPING CRITERIA> PaymentDate, PayeeName
<GROUP SORT ASCENDING> PaymentDate, PayeeName
<END DEFINE LEVEL> PaymentsByPayDatePayee
```

The five payments will generate the following four groups (instances) for the new level:

<table>
<thead>
<tr>
<th>Payment Group Instance</th>
<th>Group Criteria</th>
<th>Payments in Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group1</td>
<td>PaymentDate1, PayeeName1</td>
<td>Payment1, Payment4</td>
</tr>
<tr>
<td>Group2</td>
<td>PaymentDate1, PayeeName2</td>
<td>Payment3</td>
</tr>
<tr>
<td>Group3</td>
<td>PaymentDate1, PayeeName3</td>
<td>Payment5</td>
</tr>
<tr>
<td>Group4</td>
<td>PaymentDate2, PayeeName1</td>
<td>Payment2</td>
</tr>
</tbody>
</table>

The order of the new instances is the order that the records will print. When evaluating the multiple grouping criteria to form the instances of the new level, the criteria can be thought of as forming a hierarchy. The first criterion is at the top of the hierarchy, the last criterion is at the bottom of the hierarchy.

Generally there are two kinds of format-specific data grouping scenarios in EFT formats. Some formats print the group records only; others print the groups with the individual element records nested inside groups. Following are two examples for these scenarios based on the five payments and grouping conditions previously illustrated.

**Example**
First Scenario: Group Records Only

EFT File Structure:
- BatchRec
  - PaymentGroupHeaderRec
  - PaymentGroupFooterRec
<table>
<thead>
<tr>
<th>Record Sequence</th>
<th>Record Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BatchRec</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PaymentGroupHeaderRec</td>
<td>For group 1 (PaymentDate1, PayeeName1)</td>
</tr>
<tr>
<td>3</td>
<td>PaymentGroupFooterRec</td>
<td>For group 1 (PaymentDate1, PayeeName1)</td>
</tr>
<tr>
<td>4</td>
<td>PaymentGroupHeaderRec</td>
<td>For group 2 (PaymentDate1, PayeeName2)</td>
</tr>
<tr>
<td>5</td>
<td>PaymentGroupFooterRec</td>
<td>For group 2 (PaymentDate1, PayeeName2)</td>
</tr>
<tr>
<td>6</td>
<td>PaymentGroupHeaderRec</td>
<td>For group 3 (PaymentDate1, PayeeName3)</td>
</tr>
<tr>
<td>7</td>
<td>PaymentGroupFooterRec</td>
<td>For group 3 (PaymentDate1, PayeeName3)</td>
</tr>
<tr>
<td>8</td>
<td>PaymentGroupHeaderRec</td>
<td>For group 4 (PaymentDate2, PayeeName1)</td>
</tr>
<tr>
<td>9</td>
<td>PaymentGroupFooterRec</td>
<td>For group 4 (PaymentDate2, PayeeName1)</td>
</tr>
</tbody>
</table>

**Example**

Scenario 2: Group Records and Individual Records

EFT File Structure:

BatchRec

- PaymentGroupHeaderRec
- PaymentRec
- PaymentGroupFooterRec

Generated output:
<table>
<thead>
<tr>
<th>Record Sequence</th>
<th>Record Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BatchRec</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PaymentGroupHeaderRec</td>
<td>For group 1 (PaymentDate1, PayeeName1)</td>
</tr>
<tr>
<td>3</td>
<td>PaymentRec</td>
<td>For Payment1</td>
</tr>
<tr>
<td>4</td>
<td>PaymentRec</td>
<td>For Payment4</td>
</tr>
<tr>
<td>5</td>
<td>PaymentGroupFooterRec</td>
<td>For group 1 (PaymentDate1, PayeeName1)</td>
</tr>
<tr>
<td>6</td>
<td>PaymentGroupHeaderRec</td>
<td>For group 2 (PaymentDate1, PayeeName2)</td>
</tr>
<tr>
<td>7</td>
<td>PaymentRec</td>
<td>For Payment3</td>
</tr>
<tr>
<td>8</td>
<td>PaymentGroupFooterRec</td>
<td>For group 2 (PaymentDate1, PayeeName2)</td>
</tr>
<tr>
<td>9</td>
<td>PaymentGroupHeaderRec</td>
<td>For group 3 (PaymentDate1, PayeeName3)</td>
</tr>
<tr>
<td>10</td>
<td>PaymentRec</td>
<td>For Payment5</td>
</tr>
<tr>
<td>11</td>
<td>PaymentGroupFooterRec</td>
<td>For group 3 (PaymentDate1, PayeeName3)</td>
</tr>
<tr>
<td>12</td>
<td>PaymentGroupHeaderRec</td>
<td>For group 4 (PaymentDate2, PayeeName1)</td>
</tr>
<tr>
<td>13</td>
<td>PaymentRec</td>
<td>For Payment2</td>
</tr>
<tr>
<td>14</td>
<td>PaymentGroupFooterRec</td>
<td>For group 4 (PaymentDate2, PayeeName1)</td>
</tr>
</tbody>
</table>

Once defined with the Define Level command, the new level can be used in the template in the same manner as a level occurring in the extract. However, the records of the new level can only reference the base level fields that are defined in its grouping criteria. They cannot reference other base level fields other than in summary functions.

For example, the PaymentGroupHeaderRec can reference the PaymentDate and PayeeName in its fields. It can also reference the PaymentAmount (a payment level field) in a SUM function. However, it cannot reference other payment level fields, such as PaymentDocName or PaymentDocNum.

The Define Level command must always have one and only one grouping criteria subcommand. The Define Level command has a companion end-define level command. The subcommands must be specified between the define level and end-define level commands. They can be declared in any order.

**Define Sequence Command**

The define sequence command define a sequence that can be used in conjunction with the SEQUENCE_NUMBER function to index either the generated EFT records or the extract instances (the database records). The EFT records are the physical records defined in the template. The database records are the records from the extract. To avoid confusion, the term "record" will always refer to the EFT record. The database record will be referred to as an extract element instance or level.

The define sequence command has four subcommands: reset at level, increment basis, start at, and maximum:

**Reset at Level**

The reset at level subcommand defines where the sequence resets its starting number. It is a mandatory subcommand. For example, to number the payments in a batch, define
the reset at level as Batch. To continue numbering across batches, define the reset level as RequestHeader.

In some cases the sequence is reset outside the template. For example, a periodic sequence may be defined to reset by date. In these cases, the PERIODIC_SEQUENCE keyword is used for the reset at level. The system saves the last sequence number used for a payment file to the database. Outside events control resetting the sequence in the database. For the next payment file run, the sequence number is extracted from the database for the start at number (see start at subcommand).

Increment Basis
The increment basis subcommand specifies if the sequence should be incremented based on record or extract instances. The allowed parameters for this subcommand are RECORD and LEVEL.

Enter RECORD to increment the sequence for every record.

Enter LEVEL to increment the sequence for every new instance of a level.

Note that for levels with multiple records, if you use the level-based increment all the records in the level will have the same sequence number. The record-based increment will assign each record in the level a new sequence number.

For level-based increments, the sequence number can be used in the fields of one level only. For example, suppose an extract has a hierarchy of batch > payment > invoice and you define the increment basis by level sequence, with reset at the batch level. You can use the sequence in either the payment or invoice level fields, but not both. You cannot have sequential numbering across hierarchical levels.

However, this rule does not apply to increment basis by record sequences. Records can be sequenced across levels.

For both increment basis by level and by record sequences, the level of the sequence is implicit based on where the sequence is defined.

Define Concatenation Command
Use the define concatenation command to concatenate child-level extract elements for use in parent-level fields. For example, use this command to concatenate invoice number and due date for all the invoices belonging to a payment for use in a payment-level field.

The define concatenation command has three subcommands: base level, element, and delimiter.

Base Level Subcommand
The base level subcommand specifies the child level for the operation. For each parent-level instance, the concatenation operation loops through the child-level instances to generate the concatenated string.

Item Subcommand
The item subcommand specifies the operation used to generate each item. An item is a child-level expression that will be concatenated together to generate the concatenation string.

Delimiter Subcommand
The delimiter subcommand specifies the delimiter to separate the concatenated items in the string.
Using the SUBSTR Function

Use the SUBSTR function to break down concatenated strings into smaller strings that can be placed into different fields. For example, the following table shows five invoices in a payment:

<table>
<thead>
<tr>
<th>Invoice</th>
<th>InvoiceNum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>car_parts_inv0001</td>
</tr>
<tr>
<td>2</td>
<td>car_parts_inv0002</td>
</tr>
<tr>
<td>3</td>
<td>car_parts_inv0003</td>
</tr>
<tr>
<td>4</td>
<td>car_parts_inv0004</td>
</tr>
<tr>
<td>5</td>
<td>car_parts_inv0005</td>
</tr>
</tbody>
</table>

Using the following concatenation definition:

```
<DEFINE CONCATENATION>   ConcatenatedInvoiceInfo
<BASE LEVEL>             Invoice
<ELEMENT>                InvoiceNum
<DELIMITER>              ,
<END DEFINE CONCATENATION>ConcatenatedInvoiceInfo
```

You can reference ConcatenatedInvoiceInfo in a payment level field. The string will be:

```
car_parts_inv0001,car_parts_inv0002,car_parts_inv0003,car_parts_inv0004,car_parts_inv0005
```

If you want to use only the first forty characters of the concatenated invoice info, use either TRUNCATE function or the SUBSTR function as follows:

```
TRUNCATE(ConcatenatedInvoiceInfo, 40)
SUBSTR(ConcatenatedInvoiceInfo, 1, 40)
```

Either of these statements will result in:

```
car_parts_inv0001,car_parts_inv0002,car_`
```

To isolate the next forty characters, use the SUBSTR function:

```
SUBSTR(ConcatenatedInvoiceInfo, 41, 40)
```

to get the following string:

```
parts_inv0003,car_parts_inv0004,car_par`
```

Invalid Characters and Replacement Characters Commands

Some formats require a different character set than the one that was used to enter the data in Oracle Applications. For example, some German formats require the output file in ASCII, but the data was entered in German. If there is a mismatch between the original and target character sets you can define an ASCII equivalent to replace the original. For example, you would replace the German umlauted "a" with "ao".
Some formats will not allow certain characters. To ensure that known invalid characters will not be transmitted in your output file, use the invalid characters command to flag occurrences of specific characters.

To use the replacement characters command, specify the source characters in the left column and the replacement characters in the right column. You must enter the source characters in the original character set. This is the only case in a format template in which you use a character set not intended for output. Enter the replacement characters in the required output character set.

For DELIMITER_BASED formats, if there are delimiters in the data, you can use the escape character ”?” to retain their meaning. For example,

First name?+Last name equates to First name+Last name

Which source?? equates to Which source?

Note that the escape character itself must be escaped if it is used in data.

The replacement characters command can be used to support the escape character requirement. Specify the delimiter as the source and the escape character plus the delimiter as the target. For example, the command entry for the preceding examples would be:

```
<REPLACEMENT CHARACTERS>
+    ?+
?    ??
<END REPLACEMENT CHARACTERS>
```

The invalid character command has a single parameter that is a string of invalid characters that will cause the system to error out.

The replacement character process is performed before or during the character set conversion. The character set conversion is performed on the XML extract directly, before the formatting. After the character set conversion, the invalid characters will be checked in terms of the output character set. If no invalid characters are found, the system will proceed to formatting.

**Output Character Set and New Record Character Commands**

Use the new record character command to specify the character(s) to delimit the explicit and implicit record breaks at runtime. Each new record command represents an explicit record break. Each end of table represents an implicit record break. The parameter is a list of constant character names separated by commas.

Some formats contain no record breaks. The generated output is a single line of data. In this case, leave the new record character command parameter field empty.

**Expressions, Control Structure, and Functions**

This section describes the rules and usage for expressions in the template. It also describes supported control structures and functions.
Expressions

Expressions can be used in the data column for data fields and some command parameters. An expression is a group of XML extract fields, literals, functions, and operators. Expressions can be nested. An expression can also include the "IF" control structure. When an expression is evaluated it will always generate a result. Side effects are not allowed for the evaluation. Based on the evaluation result, expressions are classified into the following three categories:

- **Boolean Expression** - an expression that returns a boolean value, either true or false. This kind expression can be used only in the "IF-THEN-ELSE" control structure and the parameter of the display condition command.

- **Numeric Expression** - an expression that returns a number. This kind of expression can be used in numeric data fields. It can also be used in functions and commands that require numeric parameters.

- **Character Expression** - an expression that returns an alphanumeric string. This kind of expression can be used in string data fields (format type Alpha). They can also be used in functions and command that require string parameters.

Control Structures

The only supported control structure is "IF-THEN-ELSE". It can be used in an expression. The syntax is:

```plaintext
IF <boolean_expressionA> THEN
  <numeric or character expression1>
[ELSIF <boolean_expressionB THEN
  <numeric or character expression2>]
... [ELSE
  <numeric or character expression2>
END IF
```

Generally the control structure must evaluate to a number or an alphanumeric string. The control structure is considered to a numeric or character expression. The ELSIF and ELSE clauses are optional, and there can be as many ELSIF clauses as necessary. The control structure can be nested.

Functions

Following is the list of supported functions:

- **SEQUENCE_NUMBER** - is a record element index. It is used in conjunction with the Define Sequence command. It has one parameter, which is the sequence defined by the Define Sequence command. At runtime it will increase its sequence value by one each time it is referenced in a record.

- **COUNT** - counts the child level extract instances or child level records of a specific type. Declare the COUNT function on a level above the entity to be counted. The function has one argument. If the argument is a level, the function will count all the instances of the (child) level belonging to the current (parent) level instance.

For example, if the level to be counted is Payment and the current level is Batch, then the COUNT will return the total number of payments in the batch. However, if the current level is RequestHeader, the COUNT will return the total number of payments in the file across all batches. If the argument is a record type, the count function will count all the generated records of the (child level) record type belonging to the current level instance.
• INTEGER_PART, DECIMAL_PART - returns the integer or decimal portion of a numeric value. This is used in nested expressions and in commands (display condition and group by). For the final formatting of a numeric field in the data column, use the Integer/Decimal format.

• IS_NUMERIC - boolean test whether the argument is numeric. Used only with the "IF" control structure.

• TRUNCATE - truncate the first argument - a string to the length of the second argument. If the first argument is shorter than the length specified by the second argument, the first argument is returned unchanged. This is a user-friendly version for a subset of the SQL substr() functionality.

• SUM - sums all the child instance of the XML extract field argument. The field must be a numeric value. The field to be summed must always be at a lower level than the level on which the SUM function was declared.

• MIN, MAX - find the minimum or maximum of all the child instances of the XML extract field argument. The field must be a numeric value. The field to be operated on must always be at a lower level than the level on which the function was declared.

• Other SQL functions: TO_DATE, LOWER, UPPER, LENGTH, GREATEST, LEAST - use the syntax corresponding to the SQL function.

Identifiers, Operators, and Literals

This section lists the reserved key word and phrases and their usage. The supported operators are defined and the rules for referencing XML extract fields and using literals.

Key Words

There are four categories of key words and key word phrases:

• Command and column header key words
• Command parameter and function parameter key words
• Field-level key words
• Expression key words

Command and Column Header Key Words

The following key words must be used as shown: enclosed in <>s and in all capital letters with a bold font.

• <LEVEL> - the first entry of a data table. Associates the table with an XML element and specifies the hierarchy of the table.

• <END LEVEL> - declares the end of the current level. Can be used at the end of a table or in a standalone table.

• <POSITION> - column header for the first column of data field rows, which specifies the starting position of the data field in a record.

• <LENGTH> - column header for the second column of data field rows, which specifies the length of the data field.

• <FORMAT> - column header for the third column of data field rows, which specifies the data type and format setting.
• <PAD> - column header for the fourth column of data field rows, which specifies the padding style and padding character.
• <DATA> - column header for the fifth column of data field rows, which specifies the data source.
• <COMMENT> - column header for the sixth column of data field rows, which allows for free form comments.
• <NEW RECORD> - specifies a new record.
• <DISPLAY CONDITION> - specifies the condition when a record should be printed.
• <TEMPLATE TYPE> - specifies the type of the template, either FIXED_POSITION_BASED or DELIMITER_BASED.
• <OUTPUT CHARACTER SET> - specifies the character set to be used when generating the output.
• <NEW RECORD CHARACTER> - specifies the character(s) to use to signify the explicit and implicit new records at runtime.
• <DEFINE LEVEL> - defines a format-specific level in the template.
• <BASE LEVEL> - subcommand for the define level and define concatenation commands.
• <GROUPING CRITERIA> - subcommand for the define level command.
• <END DEFINE LEVEL> - signifies the end of a level.
• <DEFINE SEQUENCE> - defines a record or extract element based sequence for use in the template fields.
• <RESET AT LEVEL> - subcommand for the define sequence command.
• <INCREMENT BASIS> - subcommand for the define sequence command.
• <START AT> - subcommand for the define sequence command.
• <MAXIMUM> - subcommand for the define sequence command.
• <MAXIMUM LENGTH> - column header for the first column of data field rows, which specifies the maximum length of the data field. For DELIMITER_BASED templates only.
• <END DEFINE SEQUENCE> - signifies the end of the sequence command.
• <DEFINE CONCATENATION> - defines a concatenation of child level item that can be referenced as a string the parent level fields.
• <ELEMENT> - subcommand for the define concatenation command.
• <DELIMITER> - subcommand for the define concatenation command.
• <END DEFINE CONCATENATION> - signifies the end of the define concatenation command.
• <SORT ASCENDING> - format-specific sorting for the instances of a level.
• <SORT DESCENDING> - format-specific sorting for the instances of a level.

Command Parameter and Function Parameter Key Words
These key words must be entered in all capital letters, nonbold fonts.
• PERIODIC_SEQUENCE - used in the reset at level subcommand of the define sequence command. It denotes that the sequence number is to be reset outside the template.

• FIXED_POSITION_BASED, DELIMITER_BASED - used in the template type command, specifies the type of template.

• RECORD, LEVEL - used in the increment basis subcommand of the define sequence command. RECORD increments the sequence each time it is used in a new record. LEVEL increments the sequence only for a new instance of the level.

Field-Level Key Words

• Alpha - in the <FORMAT> column, specifies the data type is alphanumeric.

• Number - in the <FORMAT> column, specifies the data type is numeric.

• Integer - in the <FORMAT> column, used with the Number key word. Takes the integer part of the number. This has the same functionality as the INTEGER function, except the INTEGER function is used in expressions, while the Integer key word is used in the <FORMAT> column only.

• Decimal - in the <FORMAT> column, used with the Number key word. Takes the decimal part of the number. This has the same functionality as the DECIMAL function, except the DECIMAL function is used in expressions, while the Decimal key word is used in the <FORMAT> column only.

• Date - in the <FORMAT> column, specifies the data type is date.

• L, R- in the <PAD> column, specifies the side of the padding (Left or Right).

Expression Key Words

Key words and phrases used in expressions must be in capital letters and bold fonts.

• IF THEN ELSE IF THEN ELSE END IF - these key words are always used as a group. They specify the "IF" control structure expressions.

• IS NULL, IS NOT NULL - these phrases are used in the IF control structure. They form part of boolean predicates to test if an expression is NULL or not NULL.

Operators

There are two groups of operators: the boolean test operators and the expression operators. The boolean test operators include: "=", "<>", "<", ">", ">=", and "<=". They can be used only with the IF control structure. The expression operators include: "()", "||", "+", "-", and "*". They can be used in any expression.
Symbol | Usage
--- | ---
= | Equal to. Used in the IF control structure only.
<> | Not equal to. Used in the IF control structure only.
> | Greater than. Used in the IF control structure only.
< | Less than. Used in the IF control structure only.
>= | Greater than or equal to. Used in the IF control structure only.
<= | Less than or equal to. Used in the IF control structure only.
() | Function argument and expression group delimiter. The expression group inside "()" will always be evaluated first. "()" can be nested.
|| | String concatenation operator.
+ | Addition operator. Implicit type conversion may be performed if any of the operands are not numbers.
- | Subtraction operator. Implicit type conversion may be performed if any of the operands are not numbers.
* | Multiplication operator. Implicit type conversion may be performed if any of the operands are not numbers.
DIV | Division operand. Implicit type conversion may be performed if any of the operands are not numbers. Note that "/" is not used because it is part of the XPATH syntax.

Reference to XML Extract Fields and XPATH Syntax

XML elements can be used in any expression. At runtime they will be replaced with the corresponding field values. The field names are case-sensitive.

When the XML extract fields are used in the template, they must follow the XPATH syntax. This is required so that the XML Publisher engine can correctly interpret the XML elements.

There is always an extract element considered as the context element during the XML Publisher formatting process. When XML Publisher processes the data rows in a table, the level element of the table is the context element. For example, when XML Publisher processes the data rows in the Payment table, Payment is the context element. The relative XPATH you use to reference the extract elements are specified in terms of the context element.

For example if you need to refer to the PayeeName element in a Payment data table, you will specify the following relative path:
Payee/PayeeInfo/PayeeName

Each layer of the XML element hierarchy is separated by a backslash "/". You use this notation for any nested elements. The relative path for the immediate child element of the level is just the element name itself. For example, you can use TransactionID element name as is in the Payment table.

To reference a parent level element in a child level table, you can use the "../" notation. For example, in the Payment table if you need to reference the BatchName element, you can specify ../BatchName. The "../" will give you Batch as the context; in that context you can use the BatchName element name directly as BatchName is an immediate child of Batch. This notation goes up to any level for the parent elements. For example if you need to reference the RequesterParty element (in the RequestHeader) in a Payment data table, you can specify the following:

../../TrxnParties/RequesterParty

You can always use the absolute path to reference any extract element anywhere in the template. The absolute path starts with a backslash "//". For the PayeeName in the Payment table example above, you will have the following absolute path: /BatchRequest/Batch/Payment/Payee/PayeeInfo/PayeeName

The absolute path syntax provides better performance.

The identifiers defined by the setup commands such as define level, define sequence and define concatenation are considered to be global. They can be used anywhere in the template. No absolute or relative path is required. The base level and reset at level for the setup commands can also be specified. XML Publisher will be able to find the correct context for them.

If you use relative path syntax, you should specify it relative to the base levels in the following commands:

• The element subcommand of the define concatenation command
• The grouping criteria subcommand of the define level command

The extract field reference in the start at subcommand of the define sequence command should be specified with an absolute path.

The rule to reference an extract element for the level command is the same as the rule for data fields. For example, if you have a Batch level table and a nested Payment level table, you can specify the Payment element name as-is for the Payment table. Because the context for evaluating the Level command of the Payment table is the Batch.

However, if you skip the Payment level and you have an Invoice level table directly under the Batch table, you will need to specify Payment/Invoice as the level element for the Invoice table.

The XPATH syntax required by the template is very similar to UNIX/LINUX directory syntax. The context element is equivalent to the current directory. You can specify a file relative to the current directory or you can use the absolute path which starts with a "//".

Finally, the extract field reference as the result of the grouping criteria sub-command of the define level command must be specified in single quotes. This tells the XML Publisher engine to use the extract fields as the grouping criteria, not their values.
Using the Template Manager

This chapter pertains to Oracle E-Business Suite installations only.

This chapter covers the following topics:

- Introduction
- Creating the Data Definition
- Creating the Template
- Viewing and Updating a Template
- Translatable Templates

Introduction

The Template Manager is the management tool for your templates and data definitions.

**Important:** The Template Manager is available with the Oracle E-Business Suite installation only.

Use the Template Manager to:

- Register, view, and update your templates.
- Maintain data definitions for the data sources that are merged with the templates.
- Create and maintain the mapping between PDF form fields and XML elements.
- Export and upload XLIFF files for translation.
- Preview your template with sample data.

To create a template in the Template Manager:

1. Create the data definition for your template, page 5-2.
2. Register the layout template file, page 5-4.

Accessing the Template Manager

Access the Template Manager from the XML Publisher Administrator responsibility. Select **Templates** to search for or create a template. Select **Data Definitions** to search for or create a data definition.
Creating the Data Definition

When you create the data definition, you register the source of the data that will be merged with your template layout to create your published report. When you register your template layout file, you must assign it a data definition that exists in the Template Manager. This associates the two at runtime. Multiple templates can use the same data definition.

To navigate to the Create Data Definition page:

Select the Data Definitions tab, then select the Create Data Definition button.

**Name**
Enter a user-friendly name for your data definition.

**Code**
If you are using the Oracle Applications concurrent manager to generate your report, the data definition Code must match the concurrent program short name of the report program (for example, RAXCUS for the Customer Listing Summary). This enables the concurrent manager to locate the templates available for the report concurrent program when you submit the request.

**Application**
Select the report’s application from the LOV.

**Start Date**
Enter the date from which the data definition will be active.

**XML Schema**
You must supply XML Schema if both of the following conditions are applicable:
- This data definition will be assigned to a PDF template.
- The PDF template will require field mapping.
A PDF template requires mapping if the template form field names (placeholders) do not match the data element tag names of the XML file.

Use the Browse button to upload the XML Schema from a saved location.

**Note:** The W3C XML Schema Recommendation defines a standardized language for specifying the structure, content, and certain semantics of a set of XML documents. An XML schema can be considered metadata that describes a class of XML documents. The XML Schema recommendation is described at: http://www.w3.org/TR/xmlschema-0/

For more information, see Oracle XML DB Developer’s Guide 10g.

**End Date**
You cannot delete data definitions from the Template Manager. To make the data definition inactive, enter an end date.

**Preview Data**
To use the report Preview feature of the Template Manager, upload a sample XML file from the data source. The Preview feature is available from the View Template page, page 5-6 and also at runtime from the Oracle Applications request submission interface.

**Data Template**
If you are using an XML Publisher data template to generate the data for this data definition, enter the file location, or use the Browse button to upload your data template. If you are using a concurrent program to generate the data, leave this field blank. For information on creating data templates, see Data Templates, page 9-1.

After the data definition is created, all the fields are updateable except Application and Code.

**Viewing and Updating a Data Definition**
To view an existing data definition:

1. Search for the data definition from the Data Definitions tab.
2. From the search results, select the data definition Name to launch the View Data Definition page.

Access the Update Data Definition page by performing either of the following:

- Select the Update icon from the search results region.
- Select the Update button from the View Data Definition page.

From the Update Data Definition page, all fields are updateable except Application and Code. For information on the updateable fields, see Creating the Data Definition, page 5-2.

**Setting Runtime Properties for a Data Definition**
You can set runtime configuration properties that are specific to a data definition. To update or assign properties to this data definition, select the Edit Configuration button. Property values set at the Data Definition level take precedence over values set at the Site level, but will be superseded by values set at the Template level. For a full description of the properties, see Setting Configuration Properties, page 8-2.
Creating the Template

When you create a template, you assign it a data definition and upload your template layout files. Assigning the data definition makes the template available to the corresponding data source at runtime.

At initial creation, you upload one template file for a specific language and territory combination. This file will become the Default Template File (see Default Template File, page 5-5). To upload additional template files or to change the Default Template File, use the View Template page (see Viewing and Updating a Template, page 5-6).

If your template type is PDF, the Template Mapping region will display after you click the Apply button. See Template Mapping, page 5-5.

To navigate to the Create Template page:

Select the Templates tab, then select the Create Template button. To copy an existing template, see Copying a Template, page 5-6.

Name
Enter a user-friendly name for your template.

Code
Assign a template code using the product short name and a descriptive ending.

Application
Select the report's Application.

Data Definition
Select your report's data definition. The data definition must already exist in the Template Manager. To register the data definition, see Creating the Data Definition, page 5-2.
Type
Select the file type of the template. Valid template file types are: eText - Inbound, eText - Outbound, PDF, RTF, XSL-FO, XSL-HTML, XSL-TEXT, and XSL-XML.

Start Date
Enter the date from which the template will be active.

End Date
To make the template inactive, enter an end date.

Subtemplate
If this is a subtemplate, select "Yes" from the drop list.
A subtemplate is referenced by other templates, but cannot be run on its own.

File
Use the Browse button to upload your template layout file.

Language
Select the template language.

Add more language template files to your template definition from the View Template page. See Adding Templates for Additional Languages, page 5-8.

Territory
Select the language territory.

Translatable (check box)
Select this check box if you want this template to be translatable. Only RTF templates are translatable. For more information see Translatable Templates, page 5-9.

After the template definition is created, the following fields are not updateable: Application, Code, and Type. Update the template from the View Template page.

The Default Template
When you submit the XML Publisher concurrent request, you are prompted to specify the language and territory of the template that you wish to apply to the report data. If you do not select the language and territory, XML Publisher will use a template that corresponds to your session language and territory. If your session language and territory combination do not represent an available template, XML Publisher will use the Default Template to publish the report.

When you create the Template definition in the Template Manager, the original template file you upload becomes the Default Template. You can change the Default Template from the View Template page by choosing Update.

PDF Template Mapping
If your template type is PDF, the Template Mapping region displays after you select Apply. If you named the placeholders on the PDF template according to their corresponding XML element names, no mapping is required.

If you did not name the PDF placeholders according to the XML element names (or if you are using a third-party PDF template that already contained named placeholders), you must map each template field name to its corresponding XML element. You must have loaded the XML schema to the template's corresponding Data Definition to make the XML element names available to the Template Manager's mapping tool.
To perform mapping, select the **Enable Mapping** button to launch the **Update Mapping** page. See Mapping PDF Template Fields, page 5-8.

For information on creating placeholders in the PDF template, see Creating a Placeholder, page 3-5.

**Copying a Template**

Use the **Search** region to find the template you wish to copy. From the search results table, select the **Duplicate** icon for the template to launch the **Copy Template** page.

**Code**

Assign a template **Code** using the product short name and a descriptive ending.

**Name**

Enter a user-friendly name for your template.

**Application**

Select the report’s application from the LOV.

**Source Template Name**

(Not updateable) Displays the name of the template that you are duplicating.

**Viewing and Updating a Template**

Navigate to the **View Template** page:

1. Search for your template from the **Templates** page.
2. Select the template **Name** from the search results region.

**View Template: Customer Listing**

<table>
<thead>
<tr>
<th>General</th>
<th>Customer Listing Receivables</th>
<th>Code</th>
<th>Data Definition</th>
<th>Start Date</th>
<th>End Date</th>
<th>Subtemplate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Application Type Default File</td>
<td>Default File Language Default File Territory</td>
<td>Customer Listing RTF CustomerListing.rf English United States</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Template Files**

**Localized Templates**

<table>
<thead>
<tr>
<th>File Name</th>
<th>Language</th>
<th>Territory</th>
<th>Preview</th>
<th>Download</th>
<th>Update</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomerListing.rf</td>
<td>Japanese</td>
<td>Japan</td>
<td><img src="image" alt="Preview" /> <img src="image" alt="Download" /> <img src="image" alt="Update" /> <img src="image" alt="Delete" /></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Translatable Template**

<table>
<thead>
<tr>
<th>File Name</th>
<th>Language</th>
<th>Territory</th>
<th>Preview</th>
<th>Download</th>
<th>Update</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomerListing rf</td>
<td>English</td>
<td>United States</td>
<td><img src="image" alt="Preview" /> <img src="image" alt="Download" /> <img src="image" alt="Update" /> <img src="image" alt="Delete" /></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Available Translations**

- **Select Translation(s) and**... **Enable** **Disable**
- **Select All** | **Select None**

<table>
<thead>
<tr>
<th>Select</th>
<th>Status</th>
<th>Language</th>
<th>Territory</th>
<th>Preview</th>
<th>Export Translation</th>
<th>Progress</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐</td>
<td>✔</td>
<td>French</td>
<td>France</td>
<td><img src="image" alt="Preview" /> <img src="image" alt="Export Translation" /></td>
<td>✔ Complete</td>
<td></td>
</tr>
</tbody>
</table>
From the **View Template** page, you can:

- Update the general definitions, page 5-7
- Preview the template, page 5-7
- Download the template file, page 5-7
- Update the template file for editing, page 5-7
- Add localized template files for additional languages, page 5-8
- Export the XLIFF file for translation of translatable templates (RTF templates only), page 5-9
- Upload the translated XLIFF files (RTF templates only), page 5-12
- Enable or Disable available translations (RTF templates only), page 5-12
- Update the template field mapping (PDF templates only), page 5-8
- Set runtime properties for a template, page 5-9

**Note:** Seeded templates cannot be updated or deleted. The Update and Delete icons for these templates are disabled. If you wish to modify a seeded template, Duplicate, page 5-6 it, then modify the template file of the duplicated entry. You can then End Date the seeded template if you do not want it to be available to your users.

### Updating the Template General Definitions

Select the **Update** button to update the general definitions of a template. (You cannot update the **Template Code, Template Type, or Application**.) For information on the updateable fields, see Creating the Template, page 5-6.

### Previewing a Template

If you uploaded a preview data file for your data definition, the **Preview** feature will merge this data file with the selected template to allow you to immediately view a sample of the report within the Template Manager.

Select the **Preview Format** and then select the **Preview** icon next to the template file that you wish to preview. XML Publisher automatically generates a preview of your report in the format selected (PDF templates can only be viewed in PDF format).

### Editing the Template Layout

To edit the layout file of a template:

1. Select the **Download** icon to save the template file to your local file system.
2. Edit the file using your desktop application and save it in the appropriate format.
   
   For guidelines on creating template files, see Creating an RTF Template, page 2-1 or Creating a PDF Template, page 3-1.
3. Select the **Update** icon.
4. The **Add File** page prompts you to **Browse** for and select your edited file.
5. Select the **Apply** button to upload the edited file to the Template Manager.
Adding Localized Templates for Additional Languages

After you have created a template definition, you can add translated template files to support additional languages.

Use this feature when your translated template requires a different layout or adjustments to the layout. Otherwise, use the Translatable Template feature, which allows the export and upload of the translatable strings within the template. See Translatable Templates, page 5-9.

1. Select the Add File button.
2. Browse for or type in the location of the template file.
3. Select the Language for this template file from the LOV.
4. Select the Territory for this template file from the LOV.

Mapping PDF Template Fields

Select the Enable Mapping button to map the PDF template fields to the data source fields.

Update Template Definition: Project Contracts Printing Template

<table>
<thead>
<tr>
<th>Template Field Name</th>
<th>Data Source Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITEM_DESCRIPTION</td>
<td>CLASSIFIED_FLAG</td>
</tr>
<tr>
<td>LINE_NUMBER</td>
<td>L_SMALL_BUSINESS_FLAG</td>
</tr>
<tr>
<td>LINE_PROJECT_NAME</td>
<td>L_CONTACT_MAJOR_VERSION</td>
</tr>
<tr>
<td>LINE_QUANTITY</td>
<td>SCHEDULED_DELV_DEFAULT</td>
</tr>
<tr>
<td>LINE_START_DATE</td>
<td>PARTY_NAME</td>
</tr>
<tr>
<td>LINE_STATUS</td>
<td>L_CONTACT_ATTRIBUTE10</td>
</tr>
<tr>
<td>LINE_STYLE</td>
<td>L_CONTACT_ATTRIBUTE11</td>
</tr>
<tr>
<td>SUBCONTRACTED_FLAG</td>
<td>MINORITY_GROUP_LOOKUP_CODE</td>
</tr>
<tr>
<td>TEXT1</td>
<td></td>
</tr>
<tr>
<td>TEXT2</td>
<td></td>
</tr>
</tbody>
</table>

On the Update Mapping page, the Template Field Name column displays the names assigned to the form fields on the PDF template. The Data Source Element column displays a drop down list that contains all the element names from the XML schema you supplied when you created the data definition. Select the appropriate data element from the drop down list for each template field.

**Note:** Do not map the BODY_START and BODY_END grouping tags.

Once you have mapped the fields, the Update Mapping and Disable Mapping buttons become visible from the View Template page.
Setting Runtime Properties for a Template

You can set runtime configuration properties that are specific to a template. To update or assign properties to this template, select the Edit Configuration button. Property values set at the Template level take precedence over values set at the Data Definition level or at the Site Level. For a full description of the properties, see Setting Configuration Properties, page 8-2.

Translatable Templates

When you define a template as translatable, XML Publisher extracts the translatable strings. You can then export the strings into an XLIFF (.xlf) file. This XLIFF file can then be sent to a translation provider, or using a text editor, you can enter the translation for each string.

Note: XLIFF is the XML Localization Interchange File Format. It is the standard format used by localization providers. For more information about the XLIFF specification, see http://www.oasis-open.org/committees/xliff/documents/xliff-specification.htm

When translated, use the Upload Translations button to store the translated file in the Template Manager. The translated file will appear in the Available Translations region for the template.

A "translatable string" is any text in the template that is intended for display in the published report, such as table headers and field labels. Text supplied at runtime from the data is not translatable, nor is any text that you supply in the Microsoft Word form fields.

Note: Use the translatable template option when you do not require additional changes to the layout. If you wish to modify the layout for specific translated versions of your template, upload the modified, translated template as a localized template. See Adding Localized Templates for Additional Languages, page 5-8.

To define a template as translatable:
1. Select the Translatable check box from the Create Template page.

To update an existing template to be translatable:
1. Enter a Translatable File on the Update Template Definition page.

Exporting a File for Translation

The following steps summarize exporting and updating a template for translation. Editing the XLIFF file is described in further detail in the following sections.

1. Select the Export Translation button.
2. Save the .xlf file to a local directory. If your company uses a translation provider, send this file to your provider.
3. To enter your own translation, open the file with a text editor (such as WordPad).
4. The <file> element contains the attribute target-language. Replace the value of target-language with the value for the desired target language.
5. Replace the "target" element values with the desired translation for the "source" element values.

   **Caution:** Do not update the embedded data fields, page 5-11.

6. Upload the edited file to the Template Manager using the **Upload Translations** button.

Your translated file will now appear under the **Available Translations** region.

### Structure of the XLIFF File

The XLIFF file generated by XML Publisher has the following structure:

```xml
<xliff>
  <file>
    <header/>
    <body>
      <trans-unit>
        <source>
        </source>
        <target>
        </target>
        <note>
          The following figure shows an excerpt from an untranslated XLIFF file:
          
          ```xml
          <source>Text located: header/table, token &amp;1: anonymous placeholder(s)</source>
          <target>Text located: body/table</target>
          </note>
      </trans-unit>
      ...
    </body>
  </file>
</xliff>
```

**<source> and <target> Elements**

Each `<source>` element contains a translatable string from the template in the source language of the template. For example,

```xml
<source>Total</source>
```

When you initially export the XLIFF file for translation, the source and target elements are all identical. To create the translation for this template, enter the appropriate translation for each source element string in its corresponding `<target>` element.

Therefore if you were translating the sample template into German, you would enter the following for the Total string:
<source>Total</source>
<target>Gesamtbetrag</target>

**Embedded Data Fields**

Some templates contain placeholders for data fields embedded in the text display strings of the report. For example, the title of the sample report is **Italian Purchase VAT Register** - (year) where (year) is a placeholder in the RTF template that will be populated at runtime by data from an XML element. These fields are not translatable, because the value comes from the data at runtime.

To identify embedded data fields, the following token is used in the XLIFF file:

```xml
[&amp;n]
```

where \( n \) represents the numbered occurrence of a data field in the template.

For example, in the preceding XLIFF sample, the first translatable string is:

```xml
<source>Italian Purchase VAT Register - [&amp;1]</source>
```

**Warning:** Do not edit or delete the embedded data field tokens or you will affect the merging of the XML data with the template.

**source-language and target-language attributes**

The `<file>` element includes the attributes `source-language` and `target-language`. The valid value for source-language and target-language is a combination of the language code and country code as follows:

- the two-letter ISO 639 language code
- the two-letter ISO 3166 country code

For example, the value for English-United States is "en-US". This combination is also referred to as a *locale*.

When you edit the exported XLIFF file you must change the `target-language` attribute to the appropriate locale value of your target language. The following table shows examples of source-language and target-language attribute values appropriate for the given translations:

<table>
<thead>
<tr>
<th>Translation (Language/Territory)</th>
<th>source-language value</th>
<th>target-language value</th>
</tr>
</thead>
<tbody>
<tr>
<td>From English/US To English/Canadian</td>
<td>en-US</td>
<td>en-CA</td>
</tr>
<tr>
<td>From English/US To Chinese/China</td>
<td>en-US</td>
<td>zh-CN</td>
</tr>
<tr>
<td>From Japanese/Japan To French/France</td>
<td>ja-JP</td>
<td>fr-FR</td>
</tr>
</tbody>
</table>

The following figure shows the sample XLIFF file from the previous figure updated with the Chinese translation:
Uploading a Translation

To upload a translation:

1. Select the **Upload Translations** button.
2. From the **Upload Translations** page, click **Browse** to locate the translated file in your local file system, then click **Apply**.

Progress and Status Indicators

<table>
<thead>
<tr>
<th>Available Translations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select Translation(s) and...</td>
</tr>
<tr>
<td>Select All</td>
</tr>
</tbody>
</table>

- **Select**
- **Status**
- **Language**
- **Territory**
- **Preview**
- **Export Translation**
- **Progress**

<table>
<thead>
<tr>
<th>Select</th>
<th>Status</th>
<th>Language</th>
<th>Territory</th>
<th>Preview</th>
<th>Export Translation</th>
<th>Progress</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✔</td>
<td>Chinese</td>
<td>China</td>
<td>^ OK</td>
<td>^[1]</td>
<td>✔ Complete</td>
</tr>
</tbody>
</table>

When you upload a translation, if all the target elements contain data, the **Status** will be Enabled and the **Progress** will be Complete.

If XML Publisher detects that all the target elements are not populated, the **Progress** indicator displays **Incomplete**, and the **Status** defaults to Disabled.

To enable a translation, select it and click the **Enable** button. Only enabled translations are available to the Concurrent Manager. Both complete and incomplete translations can be enabled.

Updating a Translation

To update a translation file, select its **Export Translation** icon to download the XLIFF file for editing.
Generating Your Customized Report

This chapter pertains to Oracle E-Business Suite installations only.

This chapter covers the following topics:

• Using the Concurrent Manager to Generate Your Custom Output

Using the Concurrent Manager to Generate Your Custom Output

**Important:** Application Object Library (FND) patch 3435480 fully integrates XML Publisher with the Concurrent Manager. If you have taken this patch (also available in 11.5.10 Oracle E-Business Suite Cumulative Update 1), you can use the Integrated One-Step Publishing Process, page 6-1. If you have not taken this update, you must use the Two-Step Publishing Process, page 6-2.

To generate your custom output, ensure that the concurrent program is set to generate XML. A concurrent program can be set to generate XML from the Concurrent Programs window by setting the Output Format to XML:

Navigate to the Concurrent Programs window from the System Administrator or Application Developer responsibility:

• From the System Administrator responsibility, choose Concurrent, then Program, then Define.

• From the Application Developer responsibility, choose Concurrent, then Program.

Integrated One-Step Publishing Process

**Note:** Use this process if you have taken patch 3435480, Publishing Concurrent Requests with XML Publisher.

Use standard request submission to submit the report concurrent program.

• If you are using the Submit Request form, the Layout field of the Upon Completion region displays the currently selected template. To change the template, template language, or output format select the Options button.

• If you are using the HTML-based Schedule Request interface, select the template and output format from the Layout page of the process train.
Assigning a Default Template

You can assign a default template to the concurrent program that will be used by the concurrent manager and XML Publisher to publish the report unless the user selects a different template at runtime.

To assign a default template to a concurrent program:

1. Navigate to the **Update Concurrent Program** window (available from the System Administration Responsibility).
2. Select the **Onsite Setting** tab.
3. Select the template to use as the default from the **Template** list of values.

   **Note:** The Template field is not available from the Forms-based Concurrent Programs window.

Two-Step Publishing Process

**Note:** Use this process if you have not taken patch 3435480.

1. Using Standard Request Submission, submit the report, noting the request ID. The request creates the XML data file that XML Publisher will merge with the template.
2. After the request completes, use Standard Request Submission to submit the XML Publisher Concurrent Request.

   The **Parameters** window will prompt you to enter the following fields:

   - **Report Request** - Select the Request ID of the request you wish to publish.
   - **Template** - Select the template you wish to use to format the report data. Only templates registered in the Template Manager with the request data source will appear on the list.
   - **Template Locale** - Select the Language and Territory combination of the template you wish to use.

     **Note:** If you do not select a valid language and territory combination, XML Publisher will use the template that corresponds to your session language and territory. If a valid template for this combination does not exist, XML Publisher will use the Default Template. See Default Template, page 5-5.

   - **Output Format** - select the output format. If your selected template is RTF, you can generate output in Excel (HTML), HTML, PDF, or RTF. If your selected template is PDF, the output format must also be PDF.

When you submit the request, XML Publisher merges the XML data from your chosen request with the selected template to generate your selected output format.
This chapter covers the following topics:

- Extended SQL and XSL Functions
- XSL Equivalents
- Using FO Elements

### Extended SQL and XSL Functions

XML Publisher has extended a set of SQL and XSL functions for use in RTF templates. The syntax for these extended functions is

```xml
<?xdofx:expression?>
```

for extended SQL functions or

```xml
<?xdoxslt:expression?>
```

for extended XSL functions.

The supported functions are shown in the following table:

<table>
<thead>
<tr>
<th>SQL Statement</th>
<th>Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2+3</td>
<td>&lt;?xdofx:2+3?&gt;</td>
<td>Addition</td>
</tr>
<tr>
<td>2-3</td>
<td>&lt;?xdofx:2-3?&gt;</td>
<td>Subtraction</td>
</tr>
<tr>
<td>2*3</td>
<td>&lt;?xdofx:2*3?&gt;</td>
<td>Multiplication</td>
</tr>
<tr>
<td>2/3</td>
<td>&lt;?xdofx:2/3?&gt;</td>
<td>Division</td>
</tr>
<tr>
<td>2**3</td>
<td>&lt;?xdofx:2**3?&gt;</td>
<td>Exponential</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>SQL Statement</td>
<td>Usage</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| `lpad('aaa',10,'.')` | `<?xml:lpad('aaa',10,'.')?>` | The `lpad` function pads the left side of a string with a specific set of characters. The syntax for the `lpad` function is:  
  `lpad(string1,padded_length,[pad_string])`  
  `string1` is the string to pad characters to (the left-hand side).  
  `padded_length` is the number of characters to return.  
  `pad_string` is the string that will be padded to the left-hand side of `string1`. |
| `rpad('aaa',10,'.')` | `<?xml:rpad('aaa',10,'.')?>` | The `rpad` function pads the right side of a string with a specific set of characters. The syntax for the `rpad` function is:  
  `rpad(string1,padded_length,[pad_string])`  
  `string1` is the string to pad characters to (the right-hand side).  
  `padded_length` is the number of characters to return.  
  `pad_string` is the string that will be padded to the right-hand side of `string1`. |
| `decode('xxx','bbb','ccc','xxx','ddd')` | `<?xml:decode('xxx','bbb','ccc','xxx','ddd')?>` | The `decode` function has the functionality of an IF-THEN-ELSE statement. The syntax for the `decode` function is:  
  `decode(expression,search,result [,search, result]...[,default])`  
  `expression` is the value to compare.  
  `search` is the value that is compared against expression.  
  `result` is the value returned, if expression is equal to search.  
  `default` is returned if no matches are found. |
<table>
<thead>
<tr>
<th>SQL Statement</th>
<th>Usage</th>
<th>Description</th>
</tr>
</thead>
</table>
| Instr('abcabcabc', 'a', 2) | <?xdofx:Instr('abcabcabc', 'a', 2)?> | The `Instr` function returns the location of a substring in a string. The syntax for the `Instr` function is:  
  `Instr(string1, string2, [start_position], [nth_appearance])`  
  *string1* is the string to search.  
  *string2* is the substring to search for in *string1*.  
  *start_position* is the position in *string1* where the search will start. The first position in the string is 1. If the *start_position* is negative, the function counts back *start_position* number of characters from the end of *string1* and then searches towards the beginning of *string1*.  
  *nth_appearance* is the nth appearance of *string2*. |
| substr('abcdefg', 2, 3) | <?xdofx:substr('abcdefg', 2, 3)?> | The `substr` function allows you to extract a substring from a string. The syntax for the `substr` function is:  
  `substr(string, start_position, [length])`  
  *string* is the source string.  
  *start_position* is the position for extraction. The first position in the string is always 1.  
  *length* is the number of characters to extract. |
| replace(name, 'John', 'Jon') | <?xdofx:replace(name, 'John', 'Jon')?> | The `replace` function replaces a sequence of characters in a string with another set of characters. The syntax for the `replace` function is:  
  `replace(string1, string_to_replace, [replacement_string])`  
  *string1* is the string to replace a sequence of characters with another set of characters.  
  *string_to_replace* is the string that will be searched for in *string1*.  
  *replacement_string* is optional. All occurrences of *string_to_replace* will be replaced with *replacement_string* in *string1*. |
| to_number('12345') | <?xdofx:to_number('12345')?> |  
| to_char(12345) | <?xdofx:to_char(12345)?> |  
| sysdate() | <?xdofx:sysdate()?> |  
| minimum | <?xdofxslt:minimum(ELEMENT_NAME)?> | Returns the minimum value of the element in the set. |
| maximum | <?xdofxslt:maximum(ELEMENT_NAME)?> | Returns the maximum value of the element in the set. |
The following table shows supported combination functions:

<table>
<thead>
<tr>
<th>SQL Statement</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2+3/4-6*7)/8</td>
<td>&lt;?xdo fx:(2+3/4-6*7)/8?&gt;</td>
</tr>
<tr>
<td>lpad(substr('1234567890',5,3),10,'^')</td>
<td>&lt;?xdo fx:lpad(substr('1234567890',5,3),10,'^')?&gt;</td>
</tr>
<tr>
<td>decode('a','b','c','d','e','1')</td>
<td></td>
</tr>
</tbody>
</table>

**XSL Equivalents**

The following table lists the XML Publisher simplified syntax with the XSL equivalents.
### Supported XSL Elements

<table>
<thead>
<tr>
<th>Supported XSL Elements</th>
<th>Description</th>
<th>XML Publisher Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;xsl:value-of select=&quot;name&quot;&gt;</code></td>
<td>Placeholder syntax</td>
<td><code>&lt;?name?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xsl:apply-templates select=&quot;name&quot;&gt;</code></td>
<td>Applies a template rule to the current element's child nodes.</td>
<td><code>&lt;?apply:name?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xsl:copy-of select=&quot;name&quot;&gt;</code></td>
<td>Creates a copy of the current node.</td>
<td><code>&lt;?copy-of:name?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xsl:call-template name=&quot;name&quot;&gt;</code></td>
<td>Calls a named template to be inserted into/applied to the current template.</td>
<td><code>&lt;?call:name?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xsl:sort select=&quot;name&quot;&gt;</code></td>
<td>Sorts a group of data based on an element in the dataset.</td>
<td><code>&lt;?sort:name?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xsl:for-each select=&quot;name&quot;&gt;</code></td>
<td>Loops through the rows of data of a group, used to generate tabular output.</td>
<td><code>&lt;?for-each:name?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xsl:choose&gt;</code></td>
<td>Used in conjunction with when and otherwise to express multiple conditional tests.</td>
<td><code>&lt;?choose?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xsl:when test=&quot;exp&quot;&gt;</code></td>
<td>Used in conjunction with choose and otherwise to express multiple conditional tests</td>
<td><code>&lt;?when:expression?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xsl:otherwise&gt;</code></td>
<td>Used in conjunction with choose and when to express multiple conditional tests</td>
<td><code>&lt;?otherwise?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xsl:if test=&quot;exp&quot;&gt;</code></td>
<td>Used for conditional formatting.</td>
<td><code>&lt;?if:expression?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xsl:template name=&quot;name&quot;&gt;</code></td>
<td>Template declaration</td>
<td><code>&lt;?template:name?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xsl:variable name=&quot;name&quot;&gt;</code></td>
<td>Local or global variable declaration</td>
<td><code>&lt;?variable:name?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xsl:import href=&quot;url&quot;&gt;</code></td>
<td>Import the contents of one stylesheet into another</td>
<td><code>&lt;?import:url?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xsl:include href=&quot;url&quot;&gt;</code></td>
<td>Include one stylesheet in another</td>
<td><code>&lt;?include:url?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xsl:stylesheet xmlns:x=&quot;url&quot;&gt;</code></td>
<td>Define the root element of a stylesheet</td>
<td><code>&lt;?namespace:x=url?&gt;</code></td>
</tr>
</tbody>
</table>

### Using FO Elements

You can use most FO elements in an RTF template inside the Microsoft Word form fields. The following FO elements have been extended for use with XML Publisher RTF templates. The XML Publisher syntax can be used with either RTF template method.

The full list of FO elements supported by XML Publisher can be found in the Appendix: Supported XSL-FO Elements, page C-1.
<table>
<thead>
<tr>
<th>FO Element</th>
<th>XML Publisher Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;fo:page-number-citation ref-id=&quot;id&quot;</code></td>
<td><code>&lt;?fo:page-number-citation:id?</code></td>
</tr>
<tr>
<td><code>&lt;fo:page-number</code></td>
<td><code>&lt;?fo:page-number?</code></td>
</tr>
<tr>
<td><code>&lt;fo:ANY NAME WITHOUT ATTRIBUTE&gt;</code></td>
<td><code>&lt;?fo:ANY NAME WITHOUT ATTRIBUTE?&gt;</code></td>
</tr>
</tbody>
</table>
Part 2
Implementation and Developer’s Guide
This chapter covers the following topics:

- XML Publisher Administration
- Configuration
- Uploading Font Files
- Creating Font Mappings
- Locales
- Font Fallback Logic
- Font File Location
- Predefined Fonts
- Defining Currency Formats

**XML Publisher Administration**

You can customize the behavior of XML Publisher by setting properties in the Administration interface. The Administration interface allows you to:

- Set configuration properties, page 8-2
  
  **Important:** It is strongly recommended that you set a temporary directory for processing large files. If you do not, you will encounter “Out of Memory” errors. Create a temporary directory by setting a value for the Temporary directory property, page 8-3.

- Define font mappings, page 8-10
- Upload font files, page 8-10
- Define currency formats, page 8-16
The Configuration page displays all the properties grouped by type. Setting any property from this tab sets the property for the Site level. Properties can also be set at the Template level and the Data Definition level. If conflicting values are set for a property at each level, the Template level will take precedence, followed by the Data Definition level, then the Site level.

For information on setting properties at the Template level, see Setting Runtime Properties for a Template, page 5-9. For information on setting properties at the Data Definition level, see Setting Runtime Properties for a Data Definition, page 5-3.

Compatibility with the Configuration File

In previous releases of XML Publisher these properties could only be set using a configuration file (xdo.cfg). You can still use the configuration file to set these properties, and if already installed, the values will be respected. If values are entered in the Administration interface, however, these will take precedence.

The xdo configuration file must be used to set parameters specific to a server. For example, to specify different temporary directories for each server, you must use the xdo.cfg file instead of specifying it as a site-level parameter in the Administration interface.

See XML Publisher Configuration File, page A-1 for details on setting up this file.

General Properties

The property available from the General heading is:
Property Name | Internal Name | Default Value | Description
--- | --- | --- | ---
Temporary directory | system-temp-dir | N/A | Enter the directory path for the temporary directory to be used by the FO Processor when processing large files. It is strongly recommended that you set a temporary directory to avoid “Out of Memory” errors. **Note:** To set different directories for different servers, you must use the configuration file to set this property at the server level. See XML Publisher Configuration File, page A-1 for details on setting up this file.

**PDF Output Properties**

The following properties are available for PDF output:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Internal Name</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compress PDF output</td>
<td>pdf-compression</td>
<td>True</td>
<td>Specify &quot;True&quot; or &quot;False&quot; to control compression of the output PDF file.</td>
</tr>
<tr>
<td>Hide PDF viewer's menu bars</td>
<td>pdf-hide-menubar</td>
<td>False</td>
<td>Specify &quot;True&quot; to hide the viewer application’s menu bar when the document is active.</td>
</tr>
<tr>
<td>Hide PDF viewer’s toolbars</td>
<td>pdf-hide-toolbar</td>
<td>False</td>
<td>Specify &quot;True&quot; to hide the viewer application’s toolbar when the document is active.</td>
</tr>
<tr>
<td>Replace smart quotes</td>
<td>pdf_replace-smartquotes</td>
<td>True</td>
<td>Set to &quot;False&quot; if you do not want curly quotes replaced with straight quotes in your PDF output.</td>
</tr>
</tbody>
</table>

**PDF Security**

Use the following properties to control the security settings for your output PDF documents:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Internal Name</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| Enable PDF Security | pdf-security | False | If you specify "True," the output PDF file will be encrypted. You must also specify the following properties:

- Open document password
- Modify permissions password
- Encryption Level |
<p>| Open document password | pdf-open-password | N/A | This password will be required for opening the document. It will enable users to open the document only. This property is enabled only when “Enable PDF Security” is set to “True”. |
| Modify permissions password | pdf-permissions-password | N/A | This password enables users to override the security setting. This property is effective only when “Enable PDF Security” is set to “True”. |</p>
<table>
<thead>
<tr>
<th>Property Name</th>
<th>Internal Name</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encryption level</td>
<td>pdf-encryption-level</td>
<td>0 - low</td>
<td>Specify the encryption level for the output PDF file. The possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 0: Low (40-bit RC4, Acrobat 3.0 or later)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 1: High (128-bit RC4, Acrobat 5.0 or later)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This property is effective only when “Enable PDF Security” is set to “True”. When Encryption level is set to 0, you can also set the following properties:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Disable printing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Disable document modification</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Disable context copying, extraction, and accessibility</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Disable adding or changing comments and form fields</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>When Encryption level is set to 1, the following properties are available:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Enable text access for screen readers</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Enable copying of text, images, and other content</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Allowed change level</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Allowed printing level</td>
</tr>
<tr>
<td>Disable printing</td>
<td>pdf-no-printing</td>
<td>False</td>
<td>Permission available when &quot;Encryption level&quot; is set to 0. When set to &quot;True&quot;, printing is disabled for the PDF file.</td>
</tr>
<tr>
<td>Disable document modification</td>
<td>pdf-no-changing-the-document</td>
<td>False</td>
<td>Permission available when &quot;Encryption level&quot; is set to 0. When set to &quot;True&quot;, the PDF file cannot be edited.</td>
</tr>
<tr>
<td>Disable context copying, extraction, and accessibility</td>
<td>pdf-no-cceda</td>
<td>False</td>
<td>Permission available when &quot;Encryption level&quot; is set to 0. When set to &quot;True&quot;, the context copying, extraction, and accessibility features are disabled.</td>
</tr>
<tr>
<td>Disable adding or changing comments and form fields</td>
<td>pdf-no-acff</td>
<td>False</td>
<td>Permission available when &quot;Encryption level&quot; is set to 0. When set to &quot;True&quot;, the ability to add or change comments and form fields is disabled.</td>
</tr>
<tr>
<td>Enable text access for screen readers</td>
<td>pdf-enable-accessibility</td>
<td>True</td>
<td>Permission available when &quot;Encryption level&quot; is set to 1. When set to &quot;True&quot;, text access for screen reader devices is enabled.</td>
</tr>
<tr>
<td>Enable copying of text, images, and other content</td>
<td>pdf-enable-copying</td>
<td>False</td>
<td>Permission available when “Encryption level” is set to 1. When set to “True”, copying of text, images, and other content is enabled.</td>
</tr>
<tr>
<td>Property Name</td>
<td>Internal Name</td>
<td>Default Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Allowed change level</td>
<td>pdf-changes-allowed</td>
<td>0</td>
<td>Permission available when &quot;Encryption level&quot; is set to 1. Valid Values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 0: none</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 1: Allows inserting, deleting, and rotating pages</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 2: Allows filling in form fields and signing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 3: Allows commenting, filling in form fields, and signing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 4: Allows all changes except extracting pages</td>
</tr>
</tbody>
</table>

| Allowed printing level | pdf-printing-allowed | 0 | Permission available when "Encryption level" is set to 1. Valid values are: |
|                       |                      |   | • 0: None                                                                 |
|                       |                      |   | • 1: Low resolution (150 dpi)                                              |
|                       |                      |   | • 2: High resolution                                                       |

**RTF Output**

The following properties can be set to govern RTF output files:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Internal Name</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable change tracking</td>
<td>rtf-track-changes</td>
<td>False</td>
<td>Set to &quot;True&quot; to enable change tracking in the output RTF document.</td>
</tr>
<tr>
<td>Protect document for tracked changes</td>
<td>rtf-protect-document-for-tracked-changes</td>
<td>False</td>
<td>Set to &quot;True&quot; to protect the document for tracked changes.</td>
</tr>
</tbody>
</table>

**HTML Output**

The following properties can be set to govern HTML output files:
<table>
<thead>
<tr>
<th>Property Name</th>
<th>Internal Name</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base image URI</td>
<td>html-image-base-uri</td>
<td>N/A</td>
<td>Base URI which is inserted into the src attribute of the image tag before the image file name. This works only when the image is embedded in the template.</td>
</tr>
<tr>
<td>Image file directory</td>
<td>html-image-dir</td>
<td>N/A</td>
<td>Enter the directory for XML Publisher to store the image files that are embedded in the template.</td>
</tr>
<tr>
<td>Base CSS URI</td>
<td>html-css-base-uri</td>
<td>N/A</td>
<td>Base URI which is inserted into the HTML header to specify where the cascading stylesheets (CSS) for your output HTML documents will reside. You must set this property when make-accessible is true.</td>
</tr>
<tr>
<td>CSS file directory</td>
<td>html-css-dir</td>
<td>N/A</td>
<td>The CSS directory where XML Publisher stores the css file. You must set this property when make-accessible is true.</td>
</tr>
<tr>
<td>Show header</td>
<td>html-show-header</td>
<td>True</td>
<td>Set to &quot;False&quot; to suppress the template header in HTML output.</td>
</tr>
<tr>
<td>Show footer</td>
<td>html-show-footer</td>
<td>True</td>
<td>Set to &quot;False&quot; to suppress the template footer in HTML output.</td>
</tr>
<tr>
<td>Replace smart quotes</td>
<td>html-replace-smartquotes</td>
<td>True</td>
<td>Set to &quot;False&quot; if you do not want curly quotes replaced with straight quotes in your HTML output.</td>
</tr>
<tr>
<td>Character set</td>
<td>html-output-charset</td>
<td>UTF-8</td>
<td>Specify the output HTML character set.</td>
</tr>
<tr>
<td>Make HTML output</td>
<td>make-accessible</td>
<td>False</td>
<td>Specify true if you want to make the HTML output accessible.</td>
</tr>
</tbody>
</table>

**FO Processing Properties**

The following properties can be set to govern FO processing:
<table>
<thead>
<tr>
<th>Property Name</th>
<th>Internal Name</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Font mapping set</td>
<td>N/A</td>
<td>N/A</td>
<td>Select the Font Mapping Set from the list. This will be used for mapping fonts from RTF and XSL-FO templates to output PDF documents. See Creating a Font Mapping, page 8-11 for more information.</td>
</tr>
<tr>
<td>Currency format Set</td>
<td>N/A</td>
<td>N/A</td>
<td>Select the Currency Mapping Set from the list. Use a currency mapping if you want to use specific currency format masks in your templates. A currency mapping can be used for RTF and XSL-FO templates only. See Defining Currency Formats, for more information.</td>
</tr>
<tr>
<td>Bidi language digit substitution type</td>
<td>digit-substitution</td>
<td>None</td>
<td>Valid values are &quot;None&quot; and &quot;National&quot;. When set to &quot;None&quot;, Eastern European numbers will be used. When set to &quot;National&quot;, Hindi format (Arabic-Indic digits) will be used. This setting is effective only when the locale is Arabic, otherwise it is ignored.</td>
</tr>
<tr>
<td>Pages cached during processing</td>
<td>system-cache-page-size</td>
<td>50</td>
<td>This property is enabled only when you have specified a Temporary Directory (under General properties). During table of contents generation, the FO Processor caches the pages until the number of pages exceeds the value specified for this property. It then writes the pages to a file in the Temporary Directory.</td>
</tr>
<tr>
<td>Disable variable header support</td>
<td>fo-prevent-variable-header</td>
<td>False</td>
<td>If &quot;True&quot;, prevents variable header support. Variable header support automatically extends the size of the header to accommodate the contents.</td>
</tr>
<tr>
<td>Add prefix to IDs when merging FO</td>
<td>fo-merge-conflict-resolution</td>
<td>False</td>
<td>When merging multiple XSL-FO inputs, the FO Processor automatically adds random prefixes to resolve conflicting IDs. Setting this property to &quot;True&quot; disables this feature.</td>
</tr>
<tr>
<td>Use XML Publisher’s XSLT processor</td>
<td>xslt-xdoparser</td>
<td>True</td>
<td>Controls XML Publisher’s parser usage. If set to False, XSLT will not be parsed.</td>
</tr>
<tr>
<td>Enable scalable feature of XSLT processor</td>
<td>xslt-scalable</td>
<td>False</td>
<td>Controls the scalable feature of the XDO parser. The property &quot;Use XML Publisher’s XSLT processor” must be set to “True” for this property to be effective.</td>
</tr>
</tbody>
</table>

**RTF Template Properties**

The following properties can be set to govern RTF templates:
## Extract attribute sets

**Property Name:** Extract attribute sets  
**Internal Name:** rtf-extract-attribute-sets  
**Default Value:** Auto  
**Description:** The RTF processor will automatically extract attribute sets within the generated XSL-FO. The extracted sets are placed in an extra FO block, which can be referenced. This improves processing performance and reduces file size. Valid values are:

- Enable - extract attribute sets for all templates and subtemplates
- Auto - extract attribute sets for templates, but not subtemplates
- Disable - do not extract attribute sets

## Enable XPath rewriting

**Property Name:** Enable XPath rewriting  
**Internal Name:** rtf-rewrite-path  
**Default Value:** True  
**Description:** When converting an RTF template to XSL-FO, the RTF processor will automatically rewrite the XML tag names to represent the full XPath notations. Set this property to "False" to disable this feature.

## Characters used for checkbox

**Property Name:** Characters used for checkbox  
**Internal Name:** rtf-checkbox-glyph  
**Default Value:** Albany WT J;9746;9747/A  
**Description:** The XML Publisher default PDF output font does not include a glyph to represent a checkbox. If your template contains a checkbox, use this property to define a Unicode font for the representation of checkboxes in your PDF output. You must define the Unicode font number for the "checked" state and the Unicode font number for the "unchecked" state using the following syntax: `fontname;<unicode font number for true value’s glyph>;<unicode font number for false value’s glyph>`.  
**Example:** Albany WT J;9746;9747/A  
**Note:** Note that the font that you specify must be made available to XML Publisher at runtime.

## PDF Template Properties

The following properties can be set to govern PDF templates:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Font mapping set</td>
<td>N/A</td>
<td>Select the Font Mapping Set. This will be used for mapping fonts from PDF templates to output PDF documents. See Creating a Font Mapping, page 8-11 for more information.</td>
</tr>
</tbody>
</table>

## XLIFF Extraction

The following properties can be set to govern XLIFF extraction:
<table>
<thead>
<tr>
<th>Property Name</th>
<th>Internal Name</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Translation expansion percentage</td>
<td>xliff-trans-expansion</td>
<td>150 (percentage)</td>
<td>This property determines the maximum percent expansion of an extracted translation unit. For example, if set to 200, the XLIFF extractor will allow expansion by 200% - that is, a 10-character element will have a maximum width of 30 characters.</td>
</tr>
<tr>
<td>Minimum translation length</td>
<td>xliff-trans-min-length</td>
<td>15 (characters)</td>
<td>Sets a minimum length in characters for the extracted translation unit. For example, the default expansion of a 4-character field is 10 characters (based on the default setting of Translation expansion percentage of 150). If the Minimum translation length is 15, this field will be reset to 15 characters.</td>
</tr>
<tr>
<td>Maximum translation length</td>
<td>xliff-trans-max-length</td>
<td>4000 (characters)</td>
<td>Sets a limit to the calculated expansion of the translation unit (in characters). For example, the default maximum expansion of 100 characters is 250 characters. Setting Maximum translation length to 200 would limit this expansion to 200 characters.</td>
</tr>
<tr>
<td>Extract white space</td>
<td>xliff-trans-null</td>
<td>False</td>
<td>Instructs the XLIFF extractor to create a translation unit for a record that contains only spaces (is null). Set to &quot;True&quot; to generate the translation unit.</td>
</tr>
<tr>
<td>Extract sections without letters</td>
<td>xliff-trans-symbol</td>
<td>False</td>
<td>Instructs the XLIFF extractor whether to extract symbol characters. If set to &quot;False&quot; only A-Z and a-z will be extracted.</td>
</tr>
<tr>
<td>Extract words with underscores</td>
<td>xliff-trans-keyword</td>
<td>True</td>
<td>If set to &quot;False&quot;, words with underscores will not be extracted.</td>
</tr>
</tbody>
</table>
Uploading Font Files

Use the Font Files page to view and upload font files for use with XML Publisher at runtime.

To upload a font:
1. Select the Create Font File button.
2. On the Create Font File page, enter a Font Name.
3. Use the Browse button to select the font file for upload.

You can update the font file associated with a font name by selecting the Update icon from the Font Files page.

Creating Font Mappings

Use the Font Mappings page to define mappings for fonts used in your templates to desired published fonts. Font mapping is performed only for PDF output.

There are two types of mappings:
- FO to PDF - for mapping fonts from RTF templates and XSL-FO templates to PDF output fonts
- PDF Form - for mapping fonts from PDF templates to different PDF output fonts

The mapping can then be defined at the site level, the template level, or the data definition level, using the Configuration tab. See FO Processing Properties, page 8-6 for setting the FO to PDF mapping. See PDF Template Properties, page 8-8 for setting the PDF to PDF mapping.

Within a Font Mapping Set you can define multiple font mappings. Therefore you can use the same Font Mapping Set for multiple templates using different fonts, or to support multiple fonts in a single base document.
To create a Font Mapping, first create a Font Mapping Set, then create Font Mappings within the set. The Font Mapping fields will vary depending on the type of mapping you choose (FO to PDF or PDF Form).

Creating a Font Mapping Set

1. Select the Create Font Mapping Set button from the Font Mappings page.
2. On the Create Font Mapping Set page, enter a Mapping Name and Mapping Code. Enter any unique name and code you choose.
3. Select the mapping Type:
   - FO to PDF - for RTF and XSL-FO templates
   - PDF Form - for PDF templates
4. Select Apply. If there are no errors, you will receive confirmation that your mapping set was successfully created and the Font Mappings page will launch.

Creating a Font Mapping:

1. Select Create Font Mapping.
2. On the Create Font Mapping page, enter the following as appropriate and select Continue:

   If your font mapping type is FO to PDF
   Base Font
   • Font Family - enter the font family that will be mapped to a different font. For example: Arial.
   • Select the Style: Normal or Italic
   • Select the Weight: Normal or Bold

   Locale
   • (Optional) Select the Language and Territory codes. Only templates with the corresponding language and territory codes will use this font mapping. A locale is a combination of an ISO language and an ISO country. See Locales, page 8-12 for more information.

   Target Font Type
   • Select the Font Type that the base font is to be mapped to: Truetype or Type 1.

   For a list of Truetype and Type 1 fonts, see Predefined Fonts, page 8-13.

   If your font mapping type is PDF Form
   Base Font
   • Font Family - enter the font family that will be mapped to a different font. For example: serif. See Type 1 Fonts, page 8-14 for the list.

   Target Font Type
   • The target Font Type for PDF Form is always Truetype.

3. Enter the following as appropriate:
   • If you selected Truetype, or if the font mapping type is PDF Form:
     1. Select the Truetype Font from the list of fonts that have been uploaded.
2. If you want to map to a specific numbered font in the collection, enter the
TrueType Collection Number.

- If you selected Type 1, select the Font name from the list. See Type 1, page 8-14
Fonts for the list.

Once you have created your font mapping it is now available for use in your
templates. You can make this font available at one of three levels, Template, Data
Definition and Site.

For Template and Data Definition

1. Query back your template or data definition in the Template Manager and select
the Edit Configuration button.

2. Expand the FO Processing properties group and use the LOV for the Font mapping
set property to select the font mapping you want to make available for this level.

If you add the font to an individual template then only that template can use that font. If
you add it to a data definition, then all templates associated with that definition can use
the font.

For Site Level

1. Navigate to the Administration tab then select the Configuration subtab.

2. Expand the FO Processing properties group and use the LOV for the Font mapping
set property to select the font mapping you want to make available for this level.

The font will now be available across all data definitions and templates in the system.

Locales

A locale is a combination of an ISO language and an ISO country. ISO languages are
defined in ISO 639 and ISO countries are defined in ISO 3166.

The structure of the locale statement is

ISO Language-ISO country

Locales are not case-sensitive and the ISO country can be omitted.

Example locales:

- en
- en-US
- EN-US
- ja
- ko
- zh-CN

Font Fallback Logic

XML Publisher uses a font mapping fallback logic so that the result font mappings
used for a template are the a composite of the font mappings from the template up to
the site level. If a mapping is found for a font on more than one level, the most specific
level’s value overrides the others.
The resulting font mapping to use in any particular instance is the sum of all the applicable font mappings. The applicable mappings in order of preference are:

Language + Territory match, territory null > Language + Territory null (global value)

For example:

Suppose for a particular template, there are different font mapping sets assigned at the site and template levels, with the mappings shown in the following table:

<table>
<thead>
<tr>
<th>Level</th>
<th>Font Family</th>
<th>Style</th>
<th>Weight</th>
<th>Language</th>
<th>Territory</th>
<th>Target Font</th>
</tr>
</thead>
<tbody>
<tr>
<td>Site</td>
<td>Times New Roman</td>
<td>normal</td>
<td>normal</td>
<td>(none)</td>
<td>(none)</td>
<td>Times</td>
</tr>
<tr>
<td>Site</td>
<td>Arial</td>
<td>normal</td>
<td>normal</td>
<td>Japanese</td>
<td>Japan</td>
<td>Times</td>
</tr>
<tr>
<td>Template</td>
<td>Arial</td>
<td>normal</td>
<td>normal</td>
<td>Japanese</td>
<td>(none)</td>
<td>Courier</td>
</tr>
<tr>
<td>Template</td>
<td>Trebuchet MS</td>
<td>normal</td>
<td>normal</td>
<td>(none)</td>
<td>(none)</td>
<td>Helvetica</td>
</tr>
</tbody>
</table>

At runtime if the locale of the template file is Japanese/Japan, the following font mappings will be used:

<table>
<thead>
<tr>
<th>Font Family</th>
<th>Style</th>
<th>Weight</th>
<th>Target Font</th>
</tr>
</thead>
<tbody>
<tr>
<td>Times New Roman</td>
<td>normal</td>
<td>normal</td>
<td>Times</td>
</tr>
<tr>
<td>Arial</td>
<td>normal</td>
<td>normal</td>
<td>Times</td>
</tr>
<tr>
<td>Trebuchet MS</td>
<td>normal</td>
<td>normal</td>
<td>Helvetica</td>
</tr>
</tbody>
</table>

Note that even though there is a mapping for Arial at the template level, the site level value is used because it has a better match for the locale.

**Font File Location**

When using TrueType font files, the font file will be downloaded from the database to the middle-tier server before it is used by XML Publisher. The files will be placed in the XML Publisher temporary directory, in the subdirectory \{TEMP_DIR\}/xdofonts/\{environment\}

The font file will only be downloaded the first time the font is used (therefore first-time processing may be slower).

Note that if there is not a temporary directory defined, the font mechanism may produce unexpected results. See Temporary directory property, page 8-3 for information on setting the temporary directory.

**Predefined Fonts**

XML Publisher has several predefined fonts. These fonts do not require any font setting in the Administration interface.

The Type1 fonts are listed in the following table:
<table>
<thead>
<tr>
<th>Number</th>
<th>Font Family</th>
<th>Style</th>
<th>Weight</th>
<th>Font Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>serif</td>
<td>normal</td>
<td>normal</td>
<td>Time-Roman</td>
</tr>
<tr>
<td>1</td>
<td>serif</td>
<td>normal</td>
<td>bold</td>
<td>Times-Bold</td>
</tr>
<tr>
<td>1</td>
<td>serif</td>
<td>italic</td>
<td>normal</td>
<td>Times-Italic</td>
</tr>
<tr>
<td>1</td>
<td>serif</td>
<td>italic</td>
<td>bold</td>
<td>Times-BoldItalic</td>
</tr>
<tr>
<td>2</td>
<td>sans-serif</td>
<td>normal</td>
<td>normal</td>
<td>Helvetica</td>
</tr>
<tr>
<td>2</td>
<td>sans-serif</td>
<td>normal</td>
<td>bold</td>
<td>Helvetica-Bold</td>
</tr>
<tr>
<td>2</td>
<td>sans-serif</td>
<td>italic</td>
<td>normal</td>
<td>Helvetica-Oblique</td>
</tr>
<tr>
<td>2</td>
<td>sans-serif</td>
<td>italic</td>
<td>bold</td>
<td>Helvetica-BoldOblique</td>
</tr>
<tr>
<td>3</td>
<td>monospace</td>
<td>normal</td>
<td>normal</td>
<td>Courier</td>
</tr>
<tr>
<td>3</td>
<td>monospace</td>
<td>normal</td>
<td>bold</td>
<td>Courier-Bold</td>
</tr>
<tr>
<td>3</td>
<td>monospace</td>
<td>italic</td>
<td>normal</td>
<td>Courier-Oblique</td>
</tr>
<tr>
<td>3</td>
<td>monospace</td>
<td>italic</td>
<td>bold</td>
<td>Courier-BoldOblique</td>
</tr>
<tr>
<td>4</td>
<td>Courier</td>
<td>normal</td>
<td>normal</td>
<td>Courier</td>
</tr>
<tr>
<td>4</td>
<td>Courier</td>
<td>normal</td>
<td>bold</td>
<td>Courier-Bold</td>
</tr>
<tr>
<td>4</td>
<td>Courier</td>
<td>italic</td>
<td>normal</td>
<td>Courier-Oblique</td>
</tr>
<tr>
<td>4</td>
<td>Courier</td>
<td>italic</td>
<td>bold</td>
<td>Courier-BoldOblique</td>
</tr>
<tr>
<td>5</td>
<td>Helvetica</td>
<td>normal</td>
<td>normal</td>
<td>Helvetica</td>
</tr>
<tr>
<td>5</td>
<td>Helvetica</td>
<td>normal</td>
<td>bold</td>
<td>Helvetica-Bold</td>
</tr>
<tr>
<td>5</td>
<td>Helvetica</td>
<td>italic</td>
<td>normal</td>
<td>Helvetica-Oblique</td>
</tr>
<tr>
<td>5</td>
<td>Helvetica</td>
<td>italic</td>
<td>bold</td>
<td>Helvetica-BoldOblique</td>
</tr>
<tr>
<td>6</td>
<td>Times</td>
<td>normal</td>
<td>normal</td>
<td>Times</td>
</tr>
<tr>
<td>6</td>
<td>Times</td>
<td>normal</td>
<td>bold</td>
<td>Times-Bold</td>
</tr>
<tr>
<td>6</td>
<td>Times</td>
<td>italic</td>
<td>normal</td>
<td>Times-Italic</td>
</tr>
<tr>
<td>6</td>
<td>Times</td>
<td>italic</td>
<td>bold</td>
<td>Times-BoldItalic</td>
</tr>
<tr>
<td>7</td>
<td>Symbol</td>
<td>normal</td>
<td>normal</td>
<td>Symbol</td>
</tr>
<tr>
<td>8</td>
<td>ZapfDingbats</td>
<td>normal</td>
<td>normal</td>
<td>ZapfDingbats</td>
</tr>
</tbody>
</table>

The TrueType fonts are listed in the following table. All TrueType fonts will be subsetted and embedded into PDF.
<table>
<thead>
<tr>
<th>Number</th>
<th>Font Family Name</th>
<th>Style</th>
<th>Weight</th>
<th>Actual Font</th>
<th>Actual Font Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Albany WT</td>
<td>normal</td>
<td>normal</td>
<td>ALBANYWT.ttf</td>
<td>TrueType (Latin1 only)</td>
</tr>
<tr>
<td>2</td>
<td>Albany WT J</td>
<td>normal</td>
<td>normal</td>
<td>ALBANWTJ.ttf</td>
<td>TrueType (Japanese flavor)</td>
</tr>
<tr>
<td>3</td>
<td>Albany WT K</td>
<td>normal</td>
<td>normal</td>
<td>ALBANWTK.ttf</td>
<td>TrueType (Korean flavor)</td>
</tr>
<tr>
<td>4</td>
<td>Albany WT SC</td>
<td>normal</td>
<td>normal</td>
<td>ALBANWTS.ttf</td>
<td>TrueType (Simplified Chinese flavor)</td>
</tr>
<tr>
<td>5</td>
<td>Albany WT TC</td>
<td>normal</td>
<td>normal</td>
<td>ALBANWTT.ttf</td>
<td>TrueType (Traditional Chinese flavor)</td>
</tr>
<tr>
<td>6</td>
<td>Andale Duospace WT</td>
<td>normal</td>
<td>normal</td>
<td>ADUO.ttf</td>
<td>TrueType (Latin1 only, Fixed width)</td>
</tr>
<tr>
<td>6</td>
<td>Andale Duospace WT</td>
<td>bold</td>
<td>bold</td>
<td>ADUOB.ttf</td>
<td>TrueType (Latin1 only, Fixed width)</td>
</tr>
<tr>
<td>7</td>
<td>Andale Duospace WT J</td>
<td>normal</td>
<td>normal</td>
<td>ADUOJ.ttf</td>
<td>TrueType (Japanese flavor, Fixed width)</td>
</tr>
<tr>
<td>7</td>
<td>Andale Duospace WT J</td>
<td>bold</td>
<td>bold</td>
<td>ADUOJB.ttf</td>
<td>TrueType (Japanese flavor, Fixed width)</td>
</tr>
<tr>
<td>8</td>
<td>Andale Duospace WT K</td>
<td>normal</td>
<td>normal</td>
<td>ADUOK.ttf</td>
<td>TrueType (Korean flavor, Fixed width)</td>
</tr>
<tr>
<td>8</td>
<td>Andale Duospace WT K</td>
<td>bold</td>
<td>bold</td>
<td>ADUOKB.ttf</td>
<td>TrueType (Korean flavor, Fixed width)</td>
</tr>
<tr>
<td>9</td>
<td>Andale Duospace WT SC</td>
<td>normal</td>
<td>normal</td>
<td>ADUOSC.ttf</td>
<td>TrueType (Simplified Chinese flavor, Fixed width)</td>
</tr>
<tr>
<td>9</td>
<td>Andale Duospace WT SC</td>
<td>bold</td>
<td>bold</td>
<td>ADUOSCB.ttf</td>
<td>TrueType (Simplified Chinese flavor, Fixed width)</td>
</tr>
<tr>
<td>10</td>
<td>Andale Duospace WT TC</td>
<td>normal</td>
<td>normal</td>
<td>ADUOTC.ttf</td>
<td>TrueType (Traditional Chinese flavor, Fixed width)</td>
</tr>
<tr>
<td>10</td>
<td>Andale Duospace WT TC</td>
<td>bold</td>
<td>bold</td>
<td>ADUOTCB.ttf</td>
<td>TrueType (Traditional Chinese flavor, Fixed width)</td>
</tr>
</tbody>
</table>
Defining Currency Formats

The Currencies page allows you to map a number format mask to a specific currency so that your reports can display multiple currencies with their own corresponding formatting. Currency formatting is only supported for RTF and XSL-FO templates.

To utilize currency formatting, you must:
1. Define a Currency Format Set.
2. Add the specific currency format masks to the set.
3. Assign the Currency Format Set as a configuration property at the desired level (site, data definition, or template). It is available from the FO Processing Properties, page 8-6 list.
4. Enter the format-currency command in your RTF template to apply the format to the field at runtime. See Currency Formatting, page 2-101.

To define a Currency Format Set:
1. Navigate to the Currencies page under the Administration tab. Select Create Currency Format Set.
2. Enter a Name and a Code for the set. The Code is a unique identifier and cannot be changed later. Select Apply.
3. The Currency Formats page will display for your newly created set.

To add currency formats to the Currency Format Set:
1. Select Add Currency Format to add a format to your set.
2. Select a Currency Name from the list.

Note: This list is generated from the FND currency table and should include all ISO currencies. Additional currencies can be added from the System Administrator responsibility.
3. Enter the **Format Mask** you wish to use for this currency and select **Apply**.

   The Format Mask must be in the Oracle number format. The Oracle number format uses the components "9", "0", "D", and "G" to compose the format, for example: 9G999D00 where

   9 represents a displayed number only if present in data
   
   G represents the group separator
   
   D represents the decimal separator
   
   0 represents an explicitly displayed number regardless of incoming data
   
   See Using the Oracle Format Mask, page 2-94 for more information about these format mask components.

After a currency format has been created, you can update or delete it from the **Currency Formats** page.
This chapter covers the following topics:
- Introduction
- The Data Template Definition
- Constructing the Data Template
- How to Call a Data Template
- Distributed Queries
- Sample Data Templates

Introduction

The XML Publisher data engine enables you to rapidly generate any kind of XML data structure against the Oracle database in a scalable, efficient manner. The data template is the method by which you communicate your request for data to the data engine. It is an XML document whose elements collectively define how the data engine will process the template to generate the XML.

The data engine supports the following functionality:
- Schema generation
- Default RTF template generation
- Flexfields
- Single and multiple data queries
- Query links
- Parameters
- Multiple data groups
- Aggregate functions (SUM, AVG, MIN, MAX, COUNT)
- Event triggers
- Distributed queries across multiple databases

The XML output generated by the data engine supports the following:
- Unicode for XML Output
Unicode is a global character set that allows multilingual text to be displayed in a single application. This enables you to develop a single multilingual application and deploy it worldwide.

- Canonical format

  The data engine generates date elements using the canonical ISO date format: YYYY-MM-DDTHH24:MI:SS.FF3TZH:TZM for a mapped date element, and ### for number elements in the data template XML output.

The data template can be called using the Concurrent Manager or a Java API.

**Process of Overview of Implementing a Data Template for Use with the Concurrent Manager**

The process overview for implementing a data template to be called by the Concurrent Manager is as follows (this chapter covers each step in more detail):

**Using an XML or text editor:**
- Write the data template XML document following the guidelines in this chapter.

**Using the Template Manager:**
- Create a Data Definition for the data template in the Template Manager. You will upload your data template to the Template Manager.
- Register any layout templates that you wish to apply to the data generated from your data template.

**Using Oracle Applications System Administrator responsibility:**
- Register the data template as a Concurrent Program in Oracle Applications noting the following:
  - Designate “XDODTEXE” as the executable for your concurrent program. This is the XML Publisher Java concurrent program that will execute your data template.
  - The Short Name that you assign to the program must match the Data Definition Code that you assigned to the data template in the Template Manager. The XML Publisher executable uses the short name of the program to locate the corresponding data template in the Template Manager.
  - Assign the concurrent program to an appropriate Request Group for your users to run.

When your user submits the request, the Concurrent Manager executes the XML Publisher Data Template Java concurrent program. The short name of the concurrent program is used to locate the appropriate data template in the Template Manager. When the data generation is complete, the Concurrent Manager’s Output Post Processor applies the layout template to the generated XML.

**The Data Template Definition**

The data template is an XML document that consists of four basic sections: define parameters, define triggers, define data query, define data structure. This structure is shown in the following graphic:
As shown in the sample figure, the data template consists of a `<parameters>` section in which parameters are declared in child `<parameter>` elements; a `<dataQuery>` section in which the SQL queries are defined in child `<sqlStatement>` elements; and a `<dataStructure>` section in which the output XML structure is defined.

The table below lists the elements that make up the XML data template. Each element is described in detail in the following sections. Required elements are noted.

<table>
<thead>
<tr>
<th>Element</th>
<th>Attributes/Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataTemplate (Required)</td>
<td>Attributes:</td>
</tr>
<tr>
<td></td>
<td>• name (Required)</td>
</tr>
<tr>
<td></td>
<td>• description</td>
</tr>
<tr>
<td></td>
<td>• version (Required)</td>
</tr>
<tr>
<td></td>
<td>• defaultPackage - the PL/SQL package name to resolve any lexical references, group filters, or data triggers defined in the template.</td>
</tr>
<tr>
<td></td>
<td>• dataSourceRef - the default data source reference for the entire data template. Required only when performing a distributed query across multiple data sources.</td>
</tr>
<tr>
<td>parameters</td>
<td>Consists of one or more <code>&lt;parameter&gt;</code> elements.</td>
</tr>
<tr>
<td>parameter</td>
<td>Attributes:</td>
</tr>
<tr>
<td></td>
<td>• name (Required) - the parameter name that will be referenced in the template.</td>
</tr>
<tr>
<td></td>
<td>• dataType - valid values are: &quot;character&quot;, &quot;date&quot;, &quot;number&quot;</td>
</tr>
<tr>
<td></td>
<td>• defaultValue - value to use for the parameter if none supplied from the data</td>
</tr>
<tr>
<td>lexicals</td>
<td>Consists of one or more lexical elements to support flexfields.</td>
</tr>
<tr>
<td>Element</td>
<td>Attributes/Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>lexical</td>
<td>There are four types of key flexfield-related lexicals as follows:</td>
</tr>
<tr>
<td></td>
<td>• oracle.apps.fnd.flex.kff.segments_metadata</td>
</tr>
<tr>
<td></td>
<td>• oracle.apps.fnd.flex.kff.select</td>
</tr>
<tr>
<td></td>
<td>• oracle.apps.fnd.flex.kff.where</td>
</tr>
<tr>
<td></td>
<td>• oracle.apps.fnd.flex.kff.order_by</td>
</tr>
<tr>
<td>dataQuery (Required)</td>
<td>Consists of one or more <code>&lt;sqlstatement&gt;</code> elements.</td>
</tr>
<tr>
<td>sqlstatement (Required)</td>
<td>Attributes:</td>
</tr>
<tr>
<td></td>
<td>• name (Required) - the unique query identifier. Note that this name identifier will be the same across the data template. Enter the query inside the CDATA section.</td>
</tr>
<tr>
<td></td>
<td>• dataSourceRef - (for distributed queries only) specify the database against which to execute the query. If this attribute is not populated, the default data source defined in the dataTemplate element will be used.</td>
</tr>
<tr>
<td>link</td>
<td>Attributes:</td>
</tr>
<tr>
<td></td>
<td>• parentQuery - specify the parent query name.</td>
</tr>
<tr>
<td></td>
<td>• parentColumn - specify the parent column name.</td>
</tr>
<tr>
<td></td>
<td>• childQuery - specify the child query name.</td>
</tr>
<tr>
<td></td>
<td>• childColumn - specify the child column name.</td>
</tr>
<tr>
<td></td>
<td>• condition - the SQL operator that defines the relationship between the parent column and the child column. The following values for condition are supported: <code>=, &lt;, &lt;=, &gt;, &gt;=</code></td>
</tr>
<tr>
<td>dataTrigger</td>
<td>Attributes:</td>
</tr>
<tr>
<td></td>
<td>• name (Required) - the event name to fire this trigger</td>
</tr>
<tr>
<td></td>
<td>• source (Required) - the PL/SQL <code>&lt;package name&gt;.&lt;function name&gt;</code></td>
</tr>
<tr>
<td>dataStructure</td>
<td>(Required for multiple queries) Defines the structure of the output XML. Consists of <code>&lt;group&gt;</code> and <code>&lt;element&gt;</code> elements to specify the structure. This section is optional for single queries; if not specified, the data engine will generate flat XML.</td>
</tr>
<tr>
<td>Element</td>
<td>Attributes/Description</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>group</td>
<td>Consists of one or more <code>&lt;element&gt;</code> elements and sub <code>&lt;group&gt;</code> elements. Attributes: • name (Required) - the XML tag name to be assigned to the group. • source (Required) - the unique query identifier for the corresponding sql statement from which the group's elements will be derived. • groupFilter - the filter to apply to the output data group set. Define the filter as: <code>&lt;package name&gt;.&lt;function name&gt;</code>. <strong>Note:</strong> Applying a filter has performance impact. Do not use this functionality unless necessary. When possible, filter data using a WHERE clause in your query.</td>
</tr>
<tr>
<td>element (Required)</td>
<td>Attributes: • name - the tag name to assign to the element in the XML data output. • value (Required) - the column name for the SQL statement. Note that for aggregations in which the column name is in another group, the value must be defined as <code>&lt;group name&gt;.&lt;column/alias name&gt;</code>. • function - supported functions are: SUM(), COUNT(), AVG(), MIN(), MAX()</td>
</tr>
</tbody>
</table>

**Constructing the Data Template**
You can use any text or XML editor to write a data template.

**Data Template Declaration**
The `<dataTemplate>` element is the root element. It has a set of related attributes expressed within the `<dataTemplate>` tag.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>(Required) Enter the data template name.</td>
</tr>
<tr>
<td>description</td>
<td>(Optional) Enter a description of this data template.</td>
</tr>
<tr>
<td>version</td>
<td>(Required) Enter a version number for this data template.</td>
</tr>
<tr>
<td>defaultPackage</td>
<td>This attribute is required if your data template contains lexical references or any other calls to PL/SQL.</td>
</tr>
<tr>
<td>dataSourceRef</td>
<td>The default data source reference for the entire data template. Required only when performing a distributed query across multiple data sources. For more information, see Distributed Queries, page 9.25.</td>
</tr>
</tbody>
</table>
Parameters Section

A parameter is a variable whose value can be set at runtime. Parameters are especially useful for modifying SELECT statements and setting PL/SQL variables at runtime. The Parameters section of the data template is optional.

How to Define Parameters

The <parameter> element is placed between the open and close <parameters> tags. The <parameter> element has a set of related attributes. These are expressed within the <parameter> tag. For example, the name, dataType, and defaultValue attributes are expressed as follows:

```xml
<parameters>
  <parameter name="department" dataType="number" defaultValue="10"/>
</parameters>
```

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required. A keyword, unique within a given Data Template, that identifies the parameter.</td>
</tr>
<tr>
<td>dataType</td>
<td>Optional. Specify the parameter data type as &quot;character&quot;, &quot;date&quot;, or &quot;number&quot;. Default value is &quot;character&quot;. For the &quot;date&quot; dataType, the following three formats (based on the canonical ISO date format) are supported:</td>
</tr>
<tr>
<td>defaultValue</td>
<td>Optional. This value will be used for the parameter if no other value is supplied from the data at runtime.</td>
</tr>
<tr>
<td></td>
<td>• YYYY-MM-DD (example: 1997-10-24)</td>
</tr>
<tr>
<td></td>
<td>• YYYY-MM-DD HH24:MI:SS (example: 1997-10-24 12:00:00)</td>
</tr>
<tr>
<td></td>
<td>• YYYY-MM-DDTHH24:MI:SS.FF3TZH:TZM</td>
</tr>
</tbody>
</table>

How to Pass Parameters

To pass parameters, (for example, to restrict the query), use bind variables in your query. For example:

Query:

```sql
SELECT * FROM EMP
WHERE deptno=:department
```

At runtime, the value of department is passed to the query:

```sql
SELECT * FROM EMP
WHERE deptno=10
```

Data Query Section

The <dataQuery> section of the data template is required.
How to Define Queries

The `<sqlStatement>` element is placed between the open and close `dataQuery` tags. The `<sqlStatement>` element has a related attribute, `name`. It is expressed within the `<sqlStatement>` tag. The query is entered in the CDATA section. For example:

```xml
<dataQuery>
  <sqlStatement name="Q1">
    <![CDATA[SELECT DEPTNO,DNAME,LOC from dept]]>
  </sqlStatement>
</dataQuery>
```

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>A unique identifying name for the query. Note that this name will be referred to throughout the data template.</td>
</tr>
<tr>
<td>dataSourceRef</td>
<td>(For distributed queries only) Specify the database against which to execute the query. If this attribute is not populated, the default data source defined in the <code>dataTemplate</code> element will be used. For more information, see Distributed Queries, page 9-25.</td>
</tr>
</tbody>
</table>

If your column names are not unique, you must use aliases in your SELECT statements to ensure the uniqueness of your column names. If you do not use an alias, then the default column name is used. This becomes important when you specify the XML output in the `dataStructure` section. To specify an output XML element from your query you declare a `value` attribute for the element tag that corresponds to the source column.

**Tip:** Performing operations in SQL is faster than performing them in the data template or PL/SQL. It is recommended that you use SQL for the following operations:

- Use a WHERE clause instead of a group filter to exclude records.
- Perform calculations directly in your query rather than in the template.

Lexical References

You can use lexical references to replace the clauses appearing after SELECT, FROM, WHERE, GROUP BY, ORDER BY, or HAVING. Use a lexical reference when you want the parameter to replace multiple values at runtime.

Create a lexical reference using the following syntax:

```xml
&parametername
```

Define the lexical parameters as follows:

- Before creating your query, define a parameter in the PL/SQL default package for each lexical reference in the query. The data engine uses these values to replace the lexical parameters.
- Create your query containing lexical references.

For example:
Package employee
AS
  where_clause varchar2(1000);
  ....

Package body employee
AS
  ....
  where_clause := 'where deptno=10';
  ....

Data template definition:
<dataQuery>
  <sqlstatement name="Q1">
  <![CDATA[SELECT ENAME, SAL FROM EMP &where_clause]]>
  </sqlstatement>
</dataQuery>

How to Define a Data Link Between Queries

If you have multiple queries, you must link them to create the appropriate data output. In the data template, there are two methods for linking queries: using bind variables or using the <link> element to define the link between queries.

Tip: To maximize performance when building data queries in the data template:

XML Publisher tests have shown that using bind variables is more efficient than using the link tag.

The following example shows a query link using a bind variable:
<dataQuery>
  <sqlstatement name="Q1">
  <![CDATA[SELECT EMPNO, ENAME, JOB from EMP WHERE DEPTNO = :DEPTNO]]>
  </sqlstatement>
</dataQuery>

The <link> element has a set of attributes. Use these attributes to specify the required link information. You can specify any number of links. For example:
<link name="DEPTEMP_LINK" parentQuery="Q1" parentColumn="DEPTNO" childQuery="Q_2" childColumn="DEPARTMENTNO"/>
### Attribute Name | Description
---|---
**name** | Required. Enter a unique name for the link.

**parentQuery** | Specify the parent query name. This must be the name that you assigned to the corresponding `<sqlstatement>` element. See How to Define Queries, page 9-7.

**parentColumn** | Specify the parent column name.

**childQuery** | Specify the child query name. This must be the name that you assigned to the corresponding `<sqlstatement>` element. See How to Define Queries, page 9-7.

**childColumn** | Specify the child column name.

---

### Using Data Triggers

Data triggers execute PL/SQL functions at specific times during the execution and generation of XML output. Using the conditional processing capabilities of PL/SQL for these triggers, you can do things such as perform initialization tasks and access the database.

Data triggers are optional, and you can have as many `<dataTrigger>` elements as necessary.

The `<dataTrigger>` element has a set of related attributes. These are expressed within the `<dataTrigger>` tag. For example, the `name` and `source` attributes are expressed as follows:

```xml
<dataTrigger name="beforeReport" source="employee.beforeReport()"/>
<dataTrigger name="beforeReport" source="employee.beforeReport(:Parameter)"/>
```

### Attribute Name | Description
---|---
**name** | The event name to fire this trigger.

**source** | The PL/SQL `<package name>.<function name>` where the executable code resides.

The location of the trigger indicate at what point the trigger fires:

- Place a `beforeReport` trigger anywhere in your data template before the `<dataStructure>` section. A `beforeReport` trigger fires before the dataQuery is executed.

- Place an `afterReport` trigger after the `<dataStructure>` section. An `afterReport` trigger fires after you exit and after XML output has been generated.

### Data Structure Section

In the data structure section you define what the XML output will be and how it will be structured. The complete group hierarchy is available for output. You can specify all the columns within each group and break the order of those columns; you can use summaries, and placeholders to further customize within the groups. The `<dataStructure>`
section is required for multiple queries and optional for single queries. If omitted for a single query, the data engine will generate flat XML.

**Defining a Group Hierarchy**

In the data template, the `<group>` element is placed between open and close `<dataStructure>` tags. Each `<group>` has a set of related elements. You can define a group hierarchy and name the element tags for the XML output.

**Creating Break Groups**

Use a break group to produce subtotals or add placeholder columns. A break group suppresses duplicate values in sequential records. You should set an Order By clause in the SQL query to suppress duplicate values.

Assign a name to the group, and declare the source query, then specify the elements you want included in that group. When you specify the element, you assign it a name that will be used as the XML output tag name, and you declare the source column as the value. If you do not assign a name, the value (or source column name) will be used as the tag name.

For example:

```xml
<dataStructure>
  <group name="G_DEPT" source="Q1"> 
    <element name="DEPT_NUMBER" value="DEPTNO" />
    <element name="DEPT_NAME" value="DNAME"/>
  </group>
  <group name="G_EMP" source="Q2"> 
    <element name="EMPLOYEE_NUMBER" value="EMPNO" />
    <element name="NAME" value="ENAME"/> 
    <element name="JOB" value="JOB" />
  </group>
</dataStructure>
```

The following table lists the attributes for the `<group>` element tag:

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Specify any unique name for the group. This name will be used as the output XML tag name for the group.</td>
</tr>
<tr>
<td>source</td>
<td>The name of the query that provides the source data for the group. The source must come from the <code>name</code> attribute of the <code>&lt;sqlStatement&gt;</code> element.</td>
</tr>
</tbody>
</table>

The following table lists the attributes for the `<element>` element tag:

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Specify any name for the element. This name will be used as the output XML tag name for the element. The name is optional. If you do not specify a name, the source column name will be used as the XML tag name.</td>
</tr>
<tr>
<td>value</td>
<td>The name of the column that provides the source data for the element (from your query).</td>
</tr>
</tbody>
</table>
Applying Group Filters

It is strongly recommended that you use a WHERE clause instead of a group filter to exclude records from your extract. Filters enable you to conditionally remove records selected by your queries, however, this approach impacts performance. Groups can have user-created filters, using PL/SQL.

The PL/SQL function must return a boolean value (TRUE or FALSE). Depending on whether the function returns TRUE or FALSE, the current record is included or excluded from the XML data output.

For example, a sample PL/SQL function might be:

```plsql
function G_EMPFilter return boolean is
    begin
    if sal < 1000 then
        return (FALSE);
    else
        return (TRUE);
    end;
end;
```

An example of the group filter in your data template definition would be:

```xml
<group name="G_DEPT" source="Q1" groupFilter="empdata.G_EMPFilter (:DEPTSAL)"
    >
    <element name="DEPT_NUMBER" value="DEPTNO" />
    <element name="DEPT_NAME" value="DNAME" />/
    <element name="DEPTSAL" value="G_EMP.SALARY" function="SUM(
    )"/>
</group>
```

Creating a Summary Column

A summary column performs a computation on another column's data. Using the function attribute of the <element> tag, you can create the following summaries: sum, average, count, minimum, and maximum.

To create a summary column, you must define the following three attributes in the element tag:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The XML tag name to be used in the XML data output.</td>
</tr>
<tr>
<td>source</td>
<td>The name of the column that contains the data on which the summary calculation is to be performed. The source column remains unchanged.</td>
</tr>
<tr>
<td>function</td>
<td>The aggregation function to be performed. The type tells the XDO data engine how to compute the summary column values. Valid values are: SUM(), AVG(), COUNT(), MAX(), and MIN().</td>
</tr>
</tbody>
</table>

The break group determines when to reset the value of the summary column. For example:
Flexfield Support

Flexfields are defined in the data template using lexical parameters.

How to define a flexfield

1. Define the SELECT statement to use for the report data.
2. Within the SELECT statement, define each flexfield as a lexical. Use the &LEXICAL_TAG to embed flexfield related lexicals into the SELECT statement.
3. Define the flexfield-related lexicals using XML tags in the data template.

Example
<dataTemplate ...
  <parameters ... ...
  </parameters>

  <lexicals ...
    <lexical type="oracle.apps.fnd.flex.kff..."
      name="<Name of the lexical>">
      comment="<comment>"
    
  </lexical>
  ...
  </lexicals>

  <dataQuery>
    <sqlStatement ...>
      SELECT &FLEX_SELECT flex_select_alias
      FROM some_table st, code_combination_table cct
      WHERE st.some_column = 'some_condition'
      ORDER BY st.some_column, &FLEX_ORDER_BY
    </sqlStatement>
  </dataQuery>
  <dataStructure .../>

  </dataTemplate>

Flexfield Lexicals

There are four types of KFF-related lexicals. These are:

- oracle.apps.fnd.flex.kff.segments_metadata
• oracle.apps.fnd.flex.select
• oracle.apps.fnd.flex.kff.where
• oracle.apps.fnd.flex.kff.order_by

Following are descriptions of each type of KFF lexical:

oracle.apps.fnd.flex.kff.segments_metadata

Use this type of lexical to retrieve flexfield-related metadata. Using this lexical, you are not required to write PL/SQL code to retrieve this metadata. Instead, define a dummy SELECT statement, then use this lexical to get the metadata.

The XML syntax for this lexical is as follows:

```xml
<lexicals>
  <lexical
    type="oracle.apps.fnd.flex.kff.segments_metadata"
    name="Name of the lexical"
    comment="Comment"
    application_short_name="Application Short Name of the KFF"
    id_flex_code="Internal code of the KFF"
    id_flex_num="Internal number of the KFF structure"
    segments="For which segment(s) is this metadata requested?"
    show_parent_segments="Should the parent segments be listed?"
    metadata_type="Type of metadata requested"/>
</lexicals>
```

The following table lists the attributes for the segments_metadata lexical:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>application_short_name</td>
<td>(Required) The application short name of the key flexfield. For example: SQLGL.</td>
</tr>
<tr>
<td>id_flex_code</td>
<td>(Required) the internal code of the key flexfield. For example: GL#</td>
</tr>
<tr>
<td>id_flex_num</td>
<td>(Required) Internal number of the key flexfield structure. For example: 101</td>
</tr>
<tr>
<td>segments</td>
<td>(Optional) Identifies for which segments this data is requested. Default value is “ALL”. See the Oracle Applications Developer’s Guide for syntax.</td>
</tr>
<tr>
<td>show_parent_segments</td>
<td>(Optional) Valid values are &quot;Y&quot; and &quot;N&quot;. Default value is &quot;Y&quot;. If a dependent segment is displayed, the parent segment is automatically displayed, even if it is not specified as displayed in the segments attribute.</td>
</tr>
<tr>
<td>metadata_type</td>
<td>(Required) Identifies what type of metadata is requested. Valid values are:</td>
</tr>
<tr>
<td></td>
<td>above_prompt - above prompt of segment(s).</td>
</tr>
<tr>
<td></td>
<td>left_prompt - left prompt of segment(s)</td>
</tr>
</tbody>
</table>

**Example**

This example shows how to request the above_prompt of the GL Balancing Segment, and the left_prompt of the GL Account Segment.
SELECT &FLEX_GL_BALANCING_APROMPT alias_gl_balancing_aprompt, &FL
EX_GL_ACCOUNT_LPROMPT alias_gl_account_lprompt
FROM dual

<lexicals>
  <lexical type="oracle.apps.fnd.flex.kff.segments_metadata"
    name="FLEX_GL_BALANCING_APROMPT"
    comment="Comment"
    application_short_name="SQLGL"
    id_flex_code="GL#"
    id_flex_num=":P_ID_FLEX_NM"
    segments="GL_BALANCING"
    metadata_type="ABOVE_PROMPT"/>
  <lexical type="oracle.apps.fnd.flex.kff.segments_metadata"
    name="FLEX_GL_ACCOUNT+LPROMPT"
    comment="Comment"
    application_short_name="SQLGL"
    id_flex_code="GL#"
    id_flex_num=":P_ID_FLEX_NUM"
    segments="GL_ACCOUNT"
    metadata_type="LEFT_PROMPT"/>
</lexicals>

oracle.apps.fnd.flex.kff.select

This type of lexical is used in the SELECT section of the statement. It is used to retrieve and process key flexfield (kff) code combination related data based on the lexical definition.

The syntax is as follows:

<lexicals>
  <lexical type="oracle.apps.fnd.flex.kff.select"
    name="Name of the lexical"
    comment="Comment"
    application_short_name="Application Short Name of the KFF"
    id_flex_code="Internal code of the KFF"
    id_flex_num="Internal number of the KFF structure"
    multiple_id_flex_num="Are multiple structures allowed?"
    code_combination_table_alias="Code Combination Table Alias"
    segments="Segments for which this data is requested"
    show_parent_segments="Should the parent segments be listed?"
    output_type="output type"/>
</lexicals>

The following table lists the attributes for this lexical:
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>application_short_name</td>
<td>(Required) The application short name of the key flexfield. For example: SQLGL.</td>
</tr>
<tr>
<td>id_flex_code</td>
<td>(Required) the internal code of the key flexfield. For example: GL#</td>
</tr>
<tr>
<td>id_flex_num</td>
<td>(Conditionally required) Internal number of the key flexfield structure. For example: 101. Required if MULTIPLE_ID_FLEX_NUM is &quot;N&quot;.</td>
</tr>
<tr>
<td>multiple_id_flex_num</td>
<td>(Optional) Indicates whether this lexical supports multiple structures or not. Valid values are &quot;Y&quot; and &quot;N&quot;. Default is &quot;N&quot;. If set to &quot;Y&quot;, then flex will assume all structures are potentially used for data reporting and it will use &lt;code_combination_table_alias&gt;.&lt;set_defining_column_name&gt; to retrieve the structure number.</td>
</tr>
<tr>
<td>code_combination_table_alias</td>
<td>(Optional) Segment column names will be prepended with this alias.</td>
</tr>
<tr>
<td>segments</td>
<td>(Optional) Identifies for which segments this data is requested. Default value is &quot;ALL&quot;. See the Oracle Applications Developer’s Guide for syntax.</td>
</tr>
<tr>
<td>show_parent_segments</td>
<td>(Optional) Valid values are &quot;Y&quot; and &quot;N&quot;. Default value is &quot;Y&quot;. If a dependent segment is displayed, the parent segment is automatically displayed, even if it is not specified as displayed in the segments attribute.</td>
</tr>
<tr>
<td>output_type</td>
<td>(Required) Indicates what kind of output should be used as the reported value. Valid values are:</td>
</tr>
<tr>
<td></td>
<td>value - segment value as it is displayed to user.</td>
</tr>
<tr>
<td></td>
<td>padded_value - padded segment value as it is displayed to user. Number type values are padded from the left. String type values are padded on the right.</td>
</tr>
<tr>
<td>description</td>
<td>Segment value's description up to the description size defined in the segment definition.</td>
</tr>
<tr>
<td>full_description</td>
<td>Segment value's description (full size).</td>
</tr>
<tr>
<td>security</td>
<td>Returns Y if the current combination is secured against the current user, N otherwise.</td>
</tr>
</tbody>
</table>

**Example**

This example shows how to report concatenated values, concatenated descriptions, the value of the GL Balancing Segment, and the full description of the GL Balancing Segment for a single structure:

```sql
SELECT &FLEX_VALUE_ALL alias_value_all,
       &FLEX_DESCR_ALL alias_descr_all,
       &FLEX_GL_BALANCING alias_gl_balancing,
       &FLEX_GL_BALANCING_FULL_DESCR alias_gl_balancing_full_descr,
       ...
FROM gl_code_combinations gcc,
     some_other_gl_table sogt
```
WHERE gcc.chart_of_accounts_id = :p_id_flex_num
    and sogt.code_combination_id = gcc.code_combination_id
    and <more conditions on sogt>

<lexicals>
  <lexical
    type="oracle.apps.fnd.flex.kff.select"
    name="FLEX_VALUE_ALL"
    comment="Comment"
    application_short_name="SQLGL"
    id_flex_code="GL#"
    id_flex_num=":P_ID_FLEX_NUM"
    multiple_id_flex_num="N"
    code_combination_table_alias="gcc"
    segments="ALL"
    show_parent_segments="Y"
    output_type="VALUE"/>
  <lexical
    type="oracle.apps.fnd.flex.kff.select"
    name="FLEX_DESCR_ALL"
    comment="Comment"
    application_short_name="SQLGL"
    id_flex_code="GL#"
    id_flex_num=":P_ID_FLEX_NUM"
    multiple_id_flex_num="N"
    code_combination_table_alias="gcc"
    segments="ALL"
    show_parent_segments="Y"
    output_type="DESCRIPTION"/>
  <lexical
    type="oracle.apps.fnd.flex.kff.select"
    name="FLEX_GL_BALANCING"
    comment="Comment"
    application_short_name="SQLGL"
    id_flex_code="GL#"
    id_flex_num=":P_ID_FLEX_NUM"
    multiple_id_flex_num="N"
    code_combination_table_alias="gcc"
    segments="GL_BALANCING"
    show_parent_segments="N"
    output_type="VALUE"/>
  <lexical
    type="oracle.apps.fnd.flex.kff.select"
    name="FLEX_GL_BALANCING_FULL_DESCR"
    comment="Comment"
    application_short_name="SQLGL"
    id_flex_code="GL#"
    id_flex_num=":P_ID_FLEX_NUM"
    multiple_id_flex_num="N"
    code_combination_table_alias="gcc"
    segments="GL_BALANCING"
    show_parent_segments="N"
    output_type="FULL_DESCRIPTION"/>
</lexicals>
oracle.apps.fnd.flex.kff.where
This type of lexical is used in the WHERE section of the statement. It is used to modify the WHERE clause such that the SELECT statement can filter based on key flexfield segment data.

The syntax for this lexical is as follows:

```
<lexicals>
  <lexical
    type="oracle.apps.fnd.flex.kff.where"
    name="Name of the lexical"
    comment="Comment"
    application_short_name="Application Short Name of the KFF"
    id_flex_code="Internal code of the KFF"
    id_flex_num="Internal number of the KFF structure"
    code_combination_table_alias="Code Combination Table Alias"
    segments="Segments for which this data is requested"
    operator="The boolean operator to be used in the condition"
    operand1="Values to be used on the right side of the operator"
    operand2="High value for the BETWEEN operator"/>
</lexicals>
```

The attributes for this lexical are listed in the following table:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>application_short_name</td>
<td>(Required) The application short name of the key flexfield. For example: SQLGL.</td>
</tr>
<tr>
<td>id_flex_code</td>
<td>(Required) the internal code of the key flexfield. For example: GL#</td>
</tr>
<tr>
<td>id_flex_num</td>
<td>(Conditionally required) Internal number of the key flexfield structure. For example: 101. Required if MULTIPLE_ID_FLEX_NUM is &quot;N&quot;.</td>
</tr>
<tr>
<td>code_combination_table_alias</td>
<td>(Optional) Segment column names will be prepended with this alias.</td>
</tr>
<tr>
<td>segments</td>
<td>(Optional) Identifies for which segments this data is requested. Default value is &quot;ALL&quot;. See the Oracle Applications Developer's Guide for syntax.</td>
</tr>
<tr>
<td>operator</td>
<td>(Required) Valid values are: =, &lt;, &gt;, &lt;=, &gt;=, !=, &lt;&gt;,</td>
</tr>
<tr>
<td>operand1</td>
<td>(Required) Values to be used on the right side of the conditional operator.</td>
</tr>
<tr>
<td>operand2</td>
<td>(Optional) High value for the BETWEEN operator.</td>
</tr>
<tr>
<td>full_description</td>
<td>Segment value's description (full size).</td>
</tr>
<tr>
<td>security</td>
<td>Returns Y if the current combination is secured against the current user, N otherwise.</td>
</tr>
</tbody>
</table>

**Example**
This example shows a filter based on the GL Account segment and the GL Balancing Segment:
SELECT <some columns>
   FROM gl_code_combinations gcc,
       some_other_gl_table sogt
WHERE gcc.chart_of_accounts_id = :p_id_flex_num
   and sogt.code_combination_id = gcc.code_combination_id
   and &FLEX_WHERE_GL_ACCOUNT
   and &FLEX_WHERE_GL_BALANCING
   and <more conditions on sogt>

<lexicals>
<lexical
    type="oracle.apps.fnd.flex.kff.where"
    name="FLEX_WHERE_GL_ACCOUNT"
    comment="Comment"
    application_short_name="SQLGL"
    id_flex_code="GL#"
    id_flex_num=":P_ID_FLEX_NUM"
    code_combination_table_alias="gcc"
    segments="GL ACCOUNT"
    operators="=="
    operand1=":P_GL_ACCOUNT"/>
<lexical
    type="oracle.apps.fnd.flex.kff.where"
    name="FLEX_WHERE_GL_BALANCING"
    comment="Comment"
    application_short_name="SQLGL"
    id_flex_code="GL#"
    id_flex_num=":P_ID_FLEX_NUM"
    code_combination_table_alias="gcc"
    segments="GL BALANCING"
    operator="BETWEEN"
    operand1=":P_GL_BALANCING_LOW"
    operand2=":P_GL_BALANCING_HIGH"/>
</lexicals>
oracle.apps.fnd.flex.kff.order_by

This type of lexical is used in the ORDER BY section of the statement. It returns a list of column expressions so that the resulting output can be sorted by the flex segment values.

The syntax for this lexical is as follows:

<lexicals>
<lexical
    type="oracle.apps.fnd.flex.kff.order_by"
    name="Name of the lexical"
    comment="Comment"
    application_short_name="Application Short Name of the KFF"
    id_flex_code="Internal code of the KFF"
    id_flex_num="Internal number of the KFF structure"
    multiple_id_flex_num="Are multiple structures allowed?"
    code_combination_table_alias="Code Combination Table Alias"
    segments="Segment(s) for which data is requested"
    show_parent_segments="List parent segments?"/>
</lexicals>
The attributes for this lexical are listed in the following table:
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>application_short_name</td>
<td>(Required) The application short name of the key flexfield. For example: SQLGL.</td>
</tr>
<tr>
<td>id_flex_code</td>
<td>(Required) the internal code of the key flexfield. For example: GL#</td>
</tr>
<tr>
<td>id_flex_num</td>
<td>(Conditionally required) Internal number of the key flexfield structure. For example: 101. Required if MULTIPLE_ID_FLEX_NUM is &quot;N&quot;.</td>
</tr>
<tr>
<td>multiple_id_flex_num</td>
<td>(Optional) Indicates whether this lexical supports multiple structures or not. Valid values are &quot;Y&quot; and &quot;N&quot;. Default is &quot;N&quot;. If set to &quot;Y&quot;, then flex will assume all structures are potentially used for data reporting and it will use <code>&lt;code_combination_table_alias&gt;.&lt;set_defining_column_name&gt;</code> to retrieve the structure number.</td>
</tr>
<tr>
<td>code_combination_table_alias</td>
<td>(Optional) Segment column names will be prepended with this alias.</td>
</tr>
<tr>
<td>segments</td>
<td>(Optional) Identifies for which segments this data is requested. Default value is &quot;ALL&quot;. See the Oracle Applications Developer’s Guide for syntax.</td>
</tr>
<tr>
<td>show_parent_segments</td>
<td>(Optional) Valid values are &quot;Y&quot; and &quot;N&quot;. Default value is &quot;Y&quot;. If a dependent segment is displayed, the parent segment is automatically displayed, even if it is not specified as displayed in the segments attribute.</td>
</tr>
</tbody>
</table>

**Example**
The following example shows results sorted based on GL Account segment and GL Balancing segment for a single structure KFF.
SELECT <some columns>
FROM gl_code_combinations gcc,
      some_other_gl_table sogt
WHERE gcc.chart_of_accounts_id = :p_id_flex_num
      and sogt.code_combination_id = gcc.code_combination_id
      and <more conditions on sogt>
ORDER BY <some order by columns>,
         &FLEX_ORDER_BY_GL_ACCOUNT,
         &FLEX_ORDER_BY_GL_BALANCING
</lexicals>

How to Call a Data Template
There are two methods for calling the data engine to process your data template:

- Concurrent Manager
- Data Engine Java APIs

Before you can use either of these methods, you must first register your data template in the Template Manager as a Data Definition. For instructions on creating a Data Definition in the Template Manager, see Creating the Data Definition, page 5-2. You will upload your data template XML file to the Template Manager.

Using the Concurrent Manager
To use the concurrent manager to execute your data template, you must register a Concurrent Program, using the define Concurrent Programs form (shown in the following figure):
Enter the following fields in the Concurrent Programs form:

**Program**
Enter a name for the data template program.

**Short Name**
The short name you assign to the concurrent program must match the Data Definition code you assigned the Data Template in the Template Manager. This allows XML Publisher to link the report definition with the corresponding Data Definition and layout Template from the Template Manager at runtime. See Data Definition Code, page 5-2.

**Application**
Enter the application with which to associate this program.

**Executable Name**
Enter the XML Publisher data engine executable: XDODTEXE

**Output Format**
Select "XML" as the output format.

You can define parameters for the data template as you would any other concurrent program. For more information about defining concurrent programs and parameters, see Concurrent Programs window, Oracle Applications System Administrator’s Guide.

After defining the concurrent program, assign it to a request group to make it accessible to the appropriate users and responsibilities. For more information on request groups, see the Oracle Applications System Administrator’s Guide.

**Using the Data Engine Java API**
The following classes comprise the data engine utility Java API:

- oracle.apps.xdo.oa.util.DataTemplate (OA wrapper API)
• oracle.apps.xdo.dataengine.DataProcessor (Core wrapper API)

The DataProcessor class is the main class to use to execute a data template with the XML Publisher Data Engine. To use this API, you will need to instantiate this class and set parameter values for the data template, connection and output destination. Once the parameters are set, you can start processing by calling processData() method.

Note: See the OracleMetaLink note 357308.1, "About Oracle XML Publisher Release 5.6.1" for links to the Java documentation.

Example
This example provides a sample data template file, then shows an annotated Java code sample of how to call it.

The sample data template is called EmpDataTemplate.xml and is stored as /home/EmpDataTemplate.xml:

```xml
<?xml version="1.0" encoding="WINDOWS-1252" ?>
<dataTemplate name="EmpData" description="Employee Details" Version="1.0">
  <parameters>
    <parameter name="p_DeptNo" dataType="character" />
  </parameters>
  <dataQuery>
    <sqlStatement name="Q1">
      <![CDATA[
        SELECT d.DEPTNO,d.DNAME,d.LOC,EMPNO,ENAME,JOB,MGR,HIREDATE,
        SAL,nvl(COMM,0)
        FROM dept d, emp e
        WHERE d.deptno=e.deptno
        AND d.deptno = nvl(:p_DeptNo,d.deptno)
      ]]>"
    </sqlStatement>
    <dataStructure>
      <group name="G_DEPT" source="Q1">
        <element name="DEPT_NUMBER" value="DEPTNO" />
        <element name="DEPT_NAME" value="DNAME" />
        <element name="DEPTSAL" value="G_EMP.SALARY" function="SUM()" />
        <element name="LOCATION" value="LOC" />
      </group>
      <group name="G_EMP" source="Q1">
        <element name="EMPLOYEE_NUMBER" value="EMPNO" />
        <element name="NAME" value="ENAME" />
        <element name="JOB" value="JOB" />
        <element name="MANAGER" value="MGR" />
        <element name="HIREDATE" value="HIREDATE" />
        <element name="SALARY" value="SAL" />
      </group>
    </dataStructure>
  </dataQuery>
</dataTemplate>
```

The following code sample is an annotated snippet of the Java code used to process the data template by the data engine:
{  
  try {  
    // Initialization - instantiate the DataProcessor class  
    DataProcessor dataProcessor = new DataProcessor();  
    
    // Set Data Template to be executed  
    dataProcessor.setDataTemplate("/home/EmpDataTemplate.xml");  
    
    // Get Parameters - this method will return an array of the  
    // parameters in the data template  
    ArrayList parameters = dataProcessor.getParameters();  
    
    // Now we have the array list we need to iterate over  
    // the parameters and assign values to them  
    Iterator it = parameters.iterator();  
    while (it.hasNext())  
      {  
        Parameter p = (Parameter) it.next();  
        if (p.getName().equals("p_DeptNo"))  
          // Here we assign the value '10' to the p_DeptNo parameter.  
          // This could have been entered from a report submission  
          // screen or passed in from another process.  
          p.setValue(new "10");  
      }  
    
    // The parameter values now need to be assigned  
    // to the data template; there are two methods  
    // available to do this: 1. Use the setParameters  
    // method to assign the 'parameters' object to the template:  
    // dataProcessor.setParameters(parameters);  
    
    // 2. or you can assign parameter values using a hashtable.  
    Hashtable parameters = new Hashtable();  
    parameters.put("p_DeptNo","10");  
    dataProcessor.setParameters(parameters);  
    
    // Now set the jdbc connection to the database that you  
    // wish to execute the template against.  
    // This sample assumes you have already created  
    // the connection object 'jdbcConnection'  
    dataProcessor.setConnection(jdbcConnection);  
    
    // Specify the output directory and file for the data file  
    dataProcessor.setOutput("/home/EmpDetails.xml")  
    
    // Process the data template  
    dataProcessor.processData();  
  }  
  catch (Exception e)  
  {  
  }  
}  

**SQL to XML Processor**  

The data engine not only supports data generation from data templates, but it can also return data by simply passing it a SQL statement. This functionality is similar to the native database support for generating XML with the added advantage that you can retrieve huge amounts of data in a hierarchical format without sacrificing performance.
and memory consumption. You SQL statement can also contain parameters that can be given values prior to final processing.

The processor will generate XML in a ROWSET/ROW format. The tag names can be overridden using the setRowsetTag and setRowsTag methods.

The following annotated code sample shows how to use the setSQL method to pass a SQL statement to the data engine and set the element names for the generated data:

**Example**

```java
//Initialization - instantiate the DataProcessor class
DataProcessor dataProcessor = new DataProcessor();
// Set the SQL to be executed
dataProcessor.setSQL("select invoicenum, invoiceval
from invoice_table where
supplierid = :SupplID");
//Setup the SuppID value to be used
Hashtable parameters = new Hashtable();
parameters.put("SupplID ", "2000");
//Set the parameters
dataProcessor.setParameters(parameters);
//Set the db connection
dataProcessor.setConnection(jdbcConnection);
//Specify the output file name and location
dataProcessor.setOutput("/home/InvoiceDetails.xml")
//Specify the root element tag name for the generated output
dataProcessor.setRowsetTag("INVOICES");
//Specify the row element tag name for the generated output
dataProcessor.setRowsetTag("INVOICE");
//Execute the SQL
dataProcessor.processData();
```

**Other Useful Methods**

The data engine has several very useful functions that can be used to generate objects or files that can be used with the other XML Publisher APIs:

**writeDefaultLayout** – once the DataTemplate has been instantiated you can call this method to generate a default RTF template that can be used with the RTFProcessor to create an XSL template to be used with the FOProcessor. Alternatively, the default RTF can be loaded into Microsoft Word for further formatting. This method can generate either a String or Stream output.

**writeXMLSchema** - once the DataTemplate has been instantiated you can call this method to generate an XML schema representation of your data template. This is very useful if you are working with PDF templates and need to create mapping from the PDF document to your XML data.

**setScalableModeOn** – if you know you are going to return a large dataset or have a long running query you can specify that the data engine enter scalable mode. This will cause it to use the disk rather than use memory to generate the output.

**setMaxRows** – this allows you to specify a fixed number of rows to be returned by the engine. This is especially useful when you want to generate some sample data to build a layout template against.
Distributed Queries

The XML Publisher data engine allows you to perform distributed queries. This enables you to access data from multiple disparate data sources. Therefore you can extract your customer data from one database, the customer’s invoice data from another system, and bring them together into a combined hierarchical XML result set.

This section provides a usage overview and a simple example. For more detailed information and additional examples, please see OracleMetaLink note 358885.1, "Oracle XML Publisher Data Engine Examples."

Usage

You must use the Data Engine API to execute a distributed query data template. You cannot use the executable provided for use with the Concurrent Manager.

Also note that the only supported method to link distributed queries is the bind variable method (the <link> tag is not supported). See How to Define a Data Link Between Queries.

Steps to Create a Distributed Query

1. Create the data template, defining the dataSourceRef attribute of the dataTemplate element and the sqlStatement element.

   - Under the dataTemplate element, use the attribute dataSourceRef to specify the default data source reference for the entire data template.
   - Under the sqlStatement element use the attribute dataSourceRef to specify the data source to be used for the query. If this is not specified, the default data source defined in the dataTemplate element will be used.

2. Create the Java class to execute the data template.

   - Create JDBC connections to the individual databases that you want to execute your queries against. For example: Oracle, DB2, MS SQL.
   - Create a hash table to hold the connections and the values for dataSourceRef defined in your data template.
   - Pass the connections and the data template to the DataProcessor to execute the data template.

Example Distributed Query: Department and Employees

This example assumes that department data is stored in an Oracle database (referred to as ODB) and the employee data is stored in a Microsoft SQL Server database (referred to as MSDB). The sample data template will execute a query across each database and combine the results into a single data set. This is possible because the departments and employees share a common link: DepartmentID.
<!- Note the dataSourceRef attribute definition - >
<parameters>
    <parameter name="DeptID" dataType="number" defaultValue="10" />
</parameters>
<sqlStatement name="Q1">
    <!- first query does not have a dataSourceRef so the default will be used - >
    <![CDATA[
        SELECT DepartmentID, Name FROM Department where DepartmentID=:DeptID
    ]]>  
</sqlStatement>
<sqlStatement name="Q2" dataSourceRef="MSDB">
    <!- second query uses MSDB to reference the MS database - >
    <![CDATA[
        SELECT DepartmentID,EmployeeID,Title,BirthDate FROM HumanResources.employee where DepartmentID=:DepartmentID
    ]]>  
</sqlStatement>
<!- Notice this query is using the :DepartmentID bind variable from Q1 - >
</sqlStatement>
</sqlStatement>
<!- No change in the data structure just referencing Q1 and Q2 - >
</group>
</group>
</dataStructure>
</dataTemplate>

The following code sample shows this data template can be executed using the XML Publisher Java APIs:
//Initialize variables
Connection orcl=null;
Connection sqlServer=null;
try {

    //Initialization - instantiate the DataProcessor class
    DataProcessor dp = new DataProcessor();
    //Set Data Template to be executed
    dp.setDataTemplate("/home/DistributedQuery.xml");

    //Get jdbc connections to the distributed databases we wish to execute the
    //template against. Assuming you have functions to retrieve the
    orcl = getOracleConnection();
    sqlServer = getSQLServerConnection();

    //Use a hashtable to assign the connection to the dataSourceRef names
    Hashtable connections = new Hashtable();
    connections.put("ODB",orcl);
    connections.put("MSDB",sqlServer);
    //Set the jdbc connections
    dp.setDistributedConnections(connections);
    //Specify the output directory and file for the data file
    dp.setOutput("/home/DistributedQuery_data.xml");
    //Process the data template
    dp.processData();
}

The generated XML will have a hierarchical structure of department and employees retrieved from the separate databases.

**Sample Data Templates**

This section contains two sample data templates:

- Employee Listing
- General Ledger Journals Listing

The sample files are annotated to provide a better description of the components of the data template. To see more data template samples, see the XML Publisher page on Oracle Technology Network (OTN) [http://www.oracle.com/technology/products/applications/publishing/index.html]. From here you can copy and paste the samples to get you started on your own data templates.

**Employee Listing Data Template**

This template extracts employee data and department details. It has a single parameter, Department Number, that has to be populated at runtime. The data is extracted using two joined queries that use the bind variable method to join the parent (Q1) query with the child (Q2) query. It also uses the event trigger functionality using a
PL/SQL package "employee" to set the where clause on the Q1 query and to provide a group filter on the G_DEPT group.

The sample data template will generate the following XML:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<dataTemplateName>
  <LIST_G_DEPT>
    <G_DEPT>
      <DEPT_NUMBER>10</DEPT_NUMBER>
      <DEPT_NAME>ACCOUNTING</DEPT_NAME>
      <LOCATION>NEW YORK</LOCATION>
      <LIST_G_EMP>
        <G_EMP>
          <EMPLOYEE_NUMBER>7782</EMPLOYEE_NUMBER>
          <NAME>CLARK</NAME>
          <JOB>MANAGER</JOB>
          <MANAGER>7839</MANAGER>
          <HIREDATE>1981-06-09T00:00:00.000-07:00</HIREDATE>
          <SALARY>2450</SALARY>
        </G_EMP>
        <G_EMP>
          <EMPLOYEE_NUMBER>7839</EMPLOYEE_NUMBER>
          <NAME>KING</NAME>
          <JOB>PRESIDENT</JOB>
          <MANAGER/>
          <HIREDATE>1981-11-17T00:00:00.000-08:00</HIREDATE>
          <SALARY>5000</SALARY>
        </G_EMP>
        ...
      </LIST_G_EMP>
      <DEPTSAL>12750</DEPTSAL>
    </G_DEPT>
    <G_DEPT>
      <DEPT_NUMBER>20</DEPT_NUMBER>
      <DEPT_NAME>RESEARCH</DEPT_NAME>
      <LOCATION>DALLAS</LOCATION>
      <LIST_G_EMP>
        <G_EMP>
          <EMPLOYEE_NUMBER>7369</EMPLOYEE_NUMBER>
          <NAME>SMITH</NAME>
          <JOB>CLERK</JOB>
          ...
        </G_EMP>
        ...
      </LIST_G_EMP>
      <DEPTSAL>10875</DEPTSAL>
    </G_DEPT>
  </LIST_G_DEPT>
</dataTemplateName>
```

Following is the data template used to extract this data.

```xml
<?xml version="1.0" encoding="WINDOWS-1252" ?>
<dataTemplate name="Employee Listing" description="" defaultPackage="employee"
```

- The template is named, an optional description
- can be provided and the default package, if any, is identified:
- Defines a single parameter for the Department Number
  - with default of 20:
    
    ```xml
    <parameter name="p_DEPTNO" dataType="character"
    defaultValue="20"/>
    ```

- This extracts the department information based on a
  - where clause from a pl/sql package:
    
    ```sql
    <![CDATA[SELECT DEPTNO,DNAME,LOC from dept
    where &pwhereclause order by deptno]]>
    ```

- This second query extracts the employee data and joins to
  - the parent query using a bind variable, :DEPTNO
    
    ```sql
    <![CDATA[SELECT EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,nvl(COMM,0) COMM
    from EMP
    WHERE DEPTNO = :DEPTNO ]]>}
    ```

- A call is made to a before fetch trigger to set the
  - where clause variable in the department query, &pwhereclause:

- The following section specifies the XML hierarchy
  - for the returning data:
    
    ```xml
    <group name="G_DEPT" source="Q1"
    groupFilter="employee.G_DEPTFilter(:DEPT_NUMBER)"
    >
    ```

- There is a group filter placed on the DEPT group.
- This is returned from the employee.G_DEPTFilter plsql package.
- It passes the DEPT_NUMBER value ("name" attribute) rather
  - than the DEPTNO value ("value" attribute)

- This creates a summary total at the department level based
  - on the salaries at the employee level for each department:

  ```xml
  <element name="DEPTSAL" value="G_EMP.SALARY" function="SUM()" />
  ```

- The following section specifies the XML hierarchy
The PL/SQL Package:

- **This is the package specification, it declares the global**
  - **variables and functions contained therein**

  procedure BeforeReportTrigger(p_test varchar2);
  p_DEPTNO NUMBER;
pwhereclause varchar2(3200);
  function G_DEPTFilter(deptno number) return boolean;
END;
/

- **This is the package body, it contains the code for the**
  - **functions/procedures**

  create or replace package body employee as

  - **this is the event trigger called from the data template**
  - **prior to the data fetch. It sets the where clause**
  - **for the department query (Q1) based on the incoming**
  - **data template parameter**

  PROCEDURE BeforeReportTrigger IS
  begin
    IF (p_DEPTNO=10) THEN
      pwhereclause :=’DEPTNO =10’;
elif (p_DEPTNO=20) THEN
      pwhereclause:=’DEPTNO =20’;
elif (p_DEPTNO=30) THEN
      pwhereclause:=’DEPTNO =30’;
elif (p_DEPTNO=40) THEN
      pwhereclause:=’DEPTNO =20’;
else
      pwhereclause:=’1=1’;
    end if;
  end;
end;

  - **This function specifies a group filter on the Q1 group.**
  - **If the department number is 30 then the data is not returned.**

  FUNCTION G_DEPTFilter(deptno number) return boolean is
BEGIN
  if deptno = 30 then
    return FALSE;
  end if;

  RETURN TRUE;
end;
END;
/

**General Ledger Journals Data Template Example**

This data template extracts GL journals data from the E-Business Suite General Ledger schema. It is based on an existing Oracle Report that has been converted to a data template format. It follows the same format as the Employee data template but has some added functionality.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<datatemplate name="GLRGNJ" defaultPackage="GLRGNJ" version="1.0">
  <parameters>
    - Parameter declaration, these will be populated at runtime.
```

9-30  Oracle XML Publisher User’s Guide
Flexfield lexical declaration, this specifies the setup required for these flexfield functions.

The first will return the full accounting flexfield with the appropriate delimiter e.g. 01-110-6140-0000-000

The second will return 'Y' if the current combination is secured against the current user, 'N' otherwise.

The original report had an AfterParameter
- and Before report triggers

- dataTrigger name="afterParameterFormTrigger" source="GLRGNJ.afterpform"
- dataTrigger name="beforeReportTrigger" source="GLRGNJ.beforereport"
- A very complex XML hierarchy can be built with summary columns referring to lower level elements

```xml
<group name="G_SOURCE" dataType="varchar2" source="Q_MAIN">
  <element name="Source" dataType="varchar2" value="Source"/>
  <element name="SOU_SUM_ACC_DR" function="sum" dataType="number" value="G_BATCHES.B_TOTAL_DR"/>
  <element name="SOU_SUM_ACC_CR" function="sum" dataType="number" value="G_BATCHES.B_TOTAL_CR"/>
  <element name="SOU_SUM_STAT_AMT" function="sum" dataType="number" value="G_BATCHES.B_TOT_STAT_AMT"/>
</group>
<group name="G_BATCHES" dataType="varchar2" source="Q_MAIN">
  <element name="Actual_Flag" dataType="varchar2" value="Actual_Flag"/>
  <element name="Batch_Id" dataType="number" value="Batch_Id"/>
  <element name="Batch_Name" dataType="varchar2" value="Batch_Name"/>
  <element name="Batch_Eff_date" dataType="date" value="Batch_Eff_date"/>
  <element name="Journal_Type" dataType="varchar2" value="Journal_Type"/>
  <element name="Cons_Sob_Flag" dataType="varchar2" value="Cons_Sob_Flag"/>
  <element name="Batch_Type" dataType="varchar2" value="Batch_Type"/>
  <element name="Batch_Posted_Date" dataType="date" value="Batch_Posted_Date"/>
  <element name="B_TOT_DR" dataType="number" value="B_TOT_DR"/>
  <element name="B_TOTAL_DR" function="sum" dataType="number" value="G_HEADERS.H_Total_Dr"/>
  <element name="B_TOT_CR" dataType="number" value="B_TOT_CR"/>
  <element name="B_TOTAL_CR" function="sum" dataType="number" value="G_HEADERS.H_Total_Cr"/>
  <element name="B_TOT_STAT_AMT" function="sum" dataType="number" value="G_HEADERS.H_TOT_STAT_AMT"/>
</group>
<group name="G_HEADERS" dataType="varchar2" source="Q_MAIN">
  <element name="Header_id" dataType="number" value="Header_id"/>
  <element name="Header_Name" dataType="varchar2" value="Header_Name"/>
  <element name="Category" dataType="varchar2" value="Category"/>
  <element name="Header_Reference" dataType="varchar2" value="Header_Reference"/>
  <element name="Currency_Code" dataType="varchar2" value="Currency_Code"/>
  <element name="H_TOT_DR" dataType="number" value="H_TOT_DR"/>
  <element name="H_Total_Dr" function="sum" dataType="number" value="G_LINES.Line_Acc_Dr"/>
  <element name="H_TOT_CR" dataType="number" value="H_TOT_CR"/>
  <element name="H_Total_Cr" function="sum" dataType="number" value="G_LINES.Line_Acc_Cr"/>
  <element name="H_TOT_STAT_AMT" function="sum" dataType="number" value="G_LINES.Line_Stat_Amount"/>
</group>
```

Data Templates 9-33
value="G_LINES.Line_Stat_Amount"/>
<group name="G_LINES" dataType="varchar2" source="Q_MAIN"
groupFilter="GLRGNJ.g_linesgroupfilter(:G_LINES.FLEXDATA_SECURE)">
    <element name="Je_Line_Num" dataType="number"
        value="Je_Line_Num"/>
    <element name="FLEXDATA_H" dataType="varchar2"
        value="FLEXDATA_H"/>
    <element name="FLEXDATA_DSP" dataType="varchar2"
        value="FLEXDATA_DSP"/>
    <element name="Line_Description" dataType="varchar2"
        value="Line_Description"/>
    <element name="Recerence1_4" dataType="varchar2"
        value="Recerence1_4"/>
    <element name="Line_Acc_Dr" dataType="number"
        value="Line_Acc_Dr"/>
    <element name="Line_Acc_Cr" dataType="number"
        value="Line_Acc_Cr"/>
    <element name="Line_Stat_Amount" dataType="number"
        value="Line_Stat_Amount"/>
    <element name="Line_Effective_Date" dataType="date"
        value="Line_Effective_Date"/>
    <element name="FLEXDATA_SECURE" dataType="varchar2"
        value="FLEXDATA_SECURE"/>
</group>
</group>
</group>
</dataStructure>

- There is an after fetch trigger, this can be used to clean up
data or update records to report that they have been reported
</dataTrigger name="afterReportTrigger"
    source="GLRGNJ.afterreport"/>
</dataTemplate>
This chapter covers the following topics:

- Introduction
- PDF Form Processing Engine
- RTF Processor Engine
- FO Processor Engine
- PDF Document Merger
- PDF Book Binder Processor
- Document Processor Engine
- Bursting Engine
- XML Publisher Properties
- Applications Layer APIs
- Datasource APIs
- Template APIs

**Introduction**

This chapter is aimed at developers who wish to create programs or applications that interact with XML Publisher through its application programming interface. This information is meant to be used in conjunction with the Javadocs available from OracleMetaLink document 357308.1, “About Oracle XML Publisher Release 5.6.1.”

XML Publisher consists of two layers: a core layer of Java APIs and an Applications layer of APIs and UI.

- The core layer contains the main processing engines that parse templates, merge data, generate output, and deliver documents.
- The Applications layer allows the Applications developer to interact with the Template Manager on a programmatic level, which in turn interacts with the core layer.

This section assumes the reader is familiar with Java programming, XML, and XSL technologies. For the Applications layer, it is assumed the reader is familiar with the Template Manager.
XML Publisher Core APIs

XML Publisher is made up of the following core API components:

- **PDF Form Processing Engine**
  Merges a PDF template with XML data (and optional metadata) to produce PDF document output.

- **RTF Processor**
  Converts an RTF template to XSL in preparation for input to the FO Engine.

- **FO Engine**
  Merges XSL and XML to produce any of the following output formats: Excel (HTML), PDF, RTF, or HTML.

- **PDF Document Merger**
  Provides optional postprocessing of PDF files to merge documents, add page numbering, and set watermarks.

- **eText Processor**
  Converts RTF eText templates to XSL and merges the XSL with XML to produce text output for EDI and EFT transmissions.

- **Document Processor (XML APIs)**
  Provides batch processing functionality to access a single API or multiple APIs by passing a single XML file to specify template names, data sources, languages, output type, output names, and destinations.

The following diagram illustrates the template type and output type options for each core processing engine:
PDF Form Processing Engine

The PDF Form Processing Engine creates a PDF document by merging a PDF template with an XML data file. This can be done using file names, streams, or an XML data string.

As input to the PDF Processing Engine you can optionally include an XML-based Template MetaInfo (.xtm) file. This is a supplemental template to define the placement of overflow data.

The FO Processing Engine also includes utilities to provide information about your PDF template. You can:

- Retrieve a list of field names from a PDF template
- Generate the XFDF data from the PDF template
- Convert XML data into XFDF using XSLT

Merging a PDF Template with XML Data

XML data can be merged with a PDF template to produce a PDF output document in three ways:

- Using input/output file names
- Using input/output streams
- Using an input XML data string
You can optionally include a metadata XML file to describe the placement of overflow data in your template.

Merging XML Data with a PDF Template Using Input/Output File Names

Input:
- Template file name (String)
- XML file name (String)
- Metadata XML file name (String)

Output:
- PDF file name (String)

Example

```java
import oracle.apps.xdo.template.FormProcessor;
...
FormProcessor fProcessor = new FormProcessor();

fProcessor.setTemplate(args[0]); // Input File (PDF) name
fProcessor.setData(args[1]); // Input XML data file name
fProcessor.setOutput(args[2]); // Output File (PDF) name
fProcessor.setMetaInfo(args[3]); // Metadata XML File name
You can omit this setting when you do not use Metadata.

fProcessor.process();
```

Merging XML Data with a PDF Template Using Input/Output Streams

Input:
- PDF Template (Input Stream)
- XML Data (Input Stream)
- Metadata XML Data (Input Stream)

Output:
- PDF (Output Stream)
Example
import java.io.*;
import oracle.apps.xdo.template.FormProcessor;

FormProcessor fProcessor = new FormProcessor();
FileInputStream fIs = new FileInputStream(originalFilePath); // Input File
FileInputStream fIs2 = new FileInputStream(dataFilePath); // Input Data
FileInputStream fIs3 = new FileInputStream(metaData); // Metadata
FileOutputStream fOs = new FileOutputStream(newFilePath); // Output File
fProcessor.setTemplate(fIs);
fProcessor.setData(fIs2); // Input Data
fProcessor.setOutput(fOs);
fProcessor.setMetaInfo(fIs3);
fProcessor.process();
fIs.close();
fOs.close();

Merging an XML Data String with a PDF Template
Input:
- Template file name (String)
- XML data (String)
- Metadata XML file name (String)
Output:
- PDF file name (String)

Example
import oracle.apps.xdo.template.FormProcessor;

FormProcessor fProcessor = new FormProcessor();

fProcessor.setTemplate(originalFilePath); // Input File (PDF) name
fProcessor.setDataString(xmlContents); // Input XML string
fProcessor.setOutput(newFilePath); // Output File (PDF) name
fProcessor.setMetaInfo(metaXml); // Metadata XML File name
You can omit this setting when you do not use Metadata.
fProcessor.process();
Retrieving a List of Field Names

Use the FormProcessor.getFieldNames() API to retrieve the field names from a PDF template. The API returns the field names into an Enumeration object.

Input:
• PDF Template

Output:
• Enumeration Object

Example
import java.util.Enumeration;
import oracle.apps.xdo.template.FormProcessor;
FormProcessor fProcessor = new FormProcessor();
fProcessor.setTemplate(filePath); // Input File (PDF) name
Enumeration enum = fProcessor.getFieldNames();
while(enum.hasMoreElements()) {
    String formName = (String)enum.nextElement();
    System.out.println("name : " + formName + ", value : " + fProcessor.getFieldValue(formName));
}

Generating XFDF Data

XML Forms Data Format (XFDF) is a format for representing forms data and annotations in a PDF document. XFDF is the XML version of Forms Data Format (FDF), a simplified version of PDF for representing forms data and annotations. Form fields in a PDF document include edit boxes, buttons, and radio buttons.

Use this class to generate XFDF data. When you create an instance of this class, an internal XFDF tree is initialized. Use append() methods to append a FIELD element to the XFDF tree by passing a String name-value pair. You can append data as many times as you want.

This class also allows you to append XML data by calling appendXML() methods. Note that you must set the appropriate XSL stylesheet by calling setStyleSheet() method before calling appendXML() methods. You can append XML data as many times as you want.

You can retrieve the internal XFDF document at any time by calling one of the following methods: toString(), toReader(), toInputStream(), or toXMLDocument().

The following is a sample of XFDF data:
Example
<?xml version="1.0" encoding="UTF-8"?>
<xdf xmlns="http://ns.adobe.com/xfd/" xml:space="preserve">
<fields>
  <field name="TITLE">
    <value>Purchase Order</value>
  </field>
  <field name="SUPPLIER_TITLE">
    <value>Supplier</value>
  </field>
  ...
</fields>
The following code example shows how the API can be used:

Example
import oracle.apps.xdo.template.FormProcessor;
import oracle.apps.xdo.template.pdf.xfdf.XFDFObject;
...
FormProcessor fProcessor = new FormProcessor();
fProcessor.setTemplate(filePath);  // Input File (PDF) name
XFDFObject xfdfObject = new XFDFObject(fProcessor.getFieldInfo());
System.out.println(xfdfObject.toString());

Converting XML Data into XFDF Format Using XSLT
Use an XSL stylesheet to convert standard XML to the XFDF format. Following is an example of the conversion of sample XML data to XFDF:

Assume your starting XML has a ROWSET/ROW format as follows:

<ROWSET>
  <ROW num="0">
    <SUPPLIER>Supplier</SUPPLIER>
    <SUPPLIERNUMBER>Supplier Number</SUPPLIERNUMBER>
    <CURRCODE>Currency</CURRCODE>
  </ROW>
  ...
</ROWSET>

From this XML you want to generate the following XFDF format:

<fields>
  <field name="SUPPLIER1">
    <value>Supplier</value>
  </field>
  <field name="SUPPLIERNUMBER1">
    <value>Supplier Number</value>
  </field>
  <field name="CURRCODE1">
    <value>Currency</value>
  </field>
  ...
</fields>
The following XSLT will carry out the transformation:
You can then use the XFDFObject to convert XML to the XFDF format using an XSLT as follows:

**Example**

```java
import java.io.*;
import oracle.apps.xdo.template.pdf.xfdf.XFDFObject;
.
.
XFDFObject xfdfObject = new XFDFObject();

xfdfObject .setStylesheet(new BufferedInputStream(new FileInputStream(xslPath))); // XSL file name
xfdfObject .appendXML( new File(xmlPath1)); // XML data file name
xfdfObject .appendXML( new File(xmlPath2)); // XML data file name
System.out.print(xfdfObject .toString());
```
RTF Processor Engine

Generating XSL

The RTF processor engine takes an RTF template as input. The processor parses the template and creates an XSL-FO template. This can then be passed along with a data source (XML file) to the FO Engine to produce PDF, HTML, RTF, or Excel (HTML) output.

Use either input/output file names or input/output streams as shown in the following examples:

Generating XSL with Input/Output File Names
Input:
  • RTF file name (String)
Output:
  • XSL file name (String)

Example
import oracle.apps.xdo.template.FOProcessor;

public static void main(String[] args) {
  RTFProcessor rtfProcessor = new RTFProcessor(args[0]); //input template
  rtfProcessor.setOutput(args[1]); // output file
  rtfProcessor.process();
  System.exit(0);
}

Generating XSL with Input/Output Stream
Input:
  • RTF (InputStream)
Output:
  • XSL (OutputStream)
Example
import oracle.apps.xdo.template.FOProcessor;
.
.
public static void main(String[] args)
{
    FileInputStream fIs = new FileInputStream(args[0]);  // input template
    FileOutputStream fOs = new FileOutputStream(args[1]);  // output
    RTFProcessor rtfProcessor = new RTFProcessor(fIs);
    rtfProcessor.setOutput(fOs);
    rtfProcessor.process();
    // Closes input Streams output Stream
    System.exit(0);
}

FO Processor Engine

Generating Output from an XML File and an XSL File
The FO Processor Engine is XML Publisher’s implementation of the W3C XSL-FO standard. It does not represent a complete implementation of every XSL-FO component. The FO Processor can generate output in PDF, RTF, HTML, or Excel (HTML) from either of the following two inputs:
- Template (XSL) and Data (XML) combination
- FO object

Both input types can be passed as file names, streams, or in an array. Set the output format by setting the setOutputFormat method to one of the following:
- FORMAT_EXCEL
- FORMAT_HTML
- FORMAT_PDF
- FORMAT_RTF

An XSL-FO utility is also provided that creates XSL-FO from the following inputs:
- XSL file and XML file
- Two XML files and two XSL files
- Two XSL-FO files (merge)

The FO object output from the XSL-FO utility can then be used as input to the FO processor.
Major Features of the FO Processor

Bidirectional Text
XML Publisher utilizes the Unicode BiDi algorithm for BiDi layout. Based on specific values for the properties writing-mode, direction, and unicode bidi, the FO Processor supports the BiDi layout.

The writing-mode property defines how word order is supported in lines and order of lines in text. That is: right-to-left, top-to-bottom or left-to-right, top-to-bottom. The direction property determines how a string of text will be written: that is, in a specific direction, such as right-to-left or left-to-right. The unicode bidi controls and manages override behavior.

Font Fallback Mechanism
The FO Processor supports a two-level font fallback mechanism. This mechanism provides control over what default fonts to use when a specified font or glyph is not found. XML Publisher provides appropriate default fallback fonts automatically without requiring any configuration. XML Publisher also supports user-defined configuration files that specify the default fonts to use. For glyph fallback, the default mechanism will only replace the glyph and not the entire string.

For more information, see XML Publisher Configuration File, page 8-1.

Variable Header and Footer
For headers and footers that require more space than what is defined in the template, the FO Processor extends the regions and reduces the body region by the difference between the value of the page header and footer and the value of the body region margin.

Horizontal Table Break
This feature supports a "Z style" of horizontal table break. The horizontal table break is not sensitive to column span, so that if the column-spanned cells exceed the page (or area width), the FO Processor splits it and does not apply any intelligent formatting to the split cell.

The following figure shows a table that is too wide to display on one page:

The following figure shows one option of how the horizontal table break will handle the wide table. In this example, a horizontal table break is inserted after the third column.
The following figure shows another option. The table breaks after the third column, but includes the first column with each new page.

Generating Output Using File Names

The following example shows how to use the FO Processor to create an output file using file names.

Input:
- XML file name (String)
- XSL file name (String)

Output:
- Output file name (String)
import oracle.apps.xdo.template.FOPProcessor;

public static void main(String[] args)
{
    FOPProcessor processor = new FOPProcessor();
    processor.setData(args[0]); // set XML input file
    processor.setTemplate(args[1]); // set XSL input file
    processor.setOutput(args[2]); // set output file
    processor.setOutputFormat(FOPProcessor.FORMAT_PDF); // Start processing
    try
    {
        processor.generate();
    }
    catch (XDOException e)
    {
        e.printStackTrace();
        System.exit(1);
    }
    System.exit(0);
}

Generating Output Using Streams

The processor can also be used with input/output streams as shown in the following example:

Input:
- XML data (InputStream)
- XSL data (InputStream)

Output:
- Output stream (OutputStream)
Example
import java.io.InputStream;
import java.io.OutputStream;
import oracle.apps.xdo.template.FOProcessor;
.
.
public void runFOProcessor(InputStream xmlInputStream,
                          InputStream xslInputStream,
                          OutputStream pdfOutputStream)
{
    FOProcessor processor = new FOProcessor();
    processor.setData(xmlInputStream);
    processor.setTemplate(xslInputStream);
    processor.setOutput(pdfOutputStream);
    // Set output format (for PDF generation)
    processor.setOutputFormat(FOProcessor.FORMAT_PDF);
    // Start processing
    try
    {
        processor.generate();
    }
    catch (XDOException e)
    {
        e.printStackTrace();
        System.exit(1);
    }
    System.exit(0);
}

Generating Output from an Array of XSL Templates and XML Data

An array of data and template combinations can be processed to generate a single output file from the multiple inputs. The number of input data sources must match the number of templates that are to be applied to the data. For example, an input of File1.xml, File2.xml, File3.xml and File1.xsl, File2.xsl, and File3.xsl will produce a single File1_File2_File3.pdf.

Input:
- XML data (Array)
- XSL data (template) (Array)

Output:
- File Name (String)
Example
import java.io.InputStream;
import java.io.OutputStream;
import oracle.apps.xdo.template.FOProcessor;
.
.
public static void main(String[] args)
{
    String[] xmlInput = {"first.xml", "second.xml", "third.xml"};
    String[] xslInput = {"first.xsl", "second.xsl", "third.xsl"};

    FOProcessor processor = new FOProcessor();
    processor.setData(xmlInput);
    processor.setTemplate(xslInput);

    processor.setOutput("/tmp/output.pdf");   // set (PDF) output file
    processor.setOutputFormat(FOProcessor.FORMAT_PDF); processor.p
rocess();
    // Start processing
    try
    {
        processor.generate();
    }
    catch (XDOException e)
    {
        e.printStackTrace();
        System.exit(1);
    }

}

Using the XSL-FO Utility
Use the XSL-FO Utility to create an XSL-FO output file from input XML and XSL files, or to merge two XSL-FO files. Output from this utility can be used to generate your final output. See Generating Output from an XSL-FO file, page 10-18.

Creating XSL-FO from an XML File and an XSL File
Input:
• XML file
• XSL file
Output:
• XSL-FO (InputStream)
Example

```java
import oracle.apps.xdo.template.fo.util.FOUtility;
.
.
public static void main(String[] args)
{
    InputStream foStream;

    // creates XSL-FO InputStream from XML(arg[0])
    // and XSL(arg[1]) filepath String
    foStream = FOUtility.createFO(args[0], args[1]);
    if (mergedFOStream == null)
    {
        System.out.println("Merge failed.");
        System.exit(1);
    }
    System.exit(0);
}
```

Creating XSL-FO from Two XML Files and Two XSL files

Input:
- XML File 1
- XML File 2
- XSL File 1
- XSL File 2

Output:
- XSL-FO (InputStream)
Example
import oracle.apps.xdo.template.fo.util.FOUtility;
.
.
public static void main(String[] args)
{
    InputStream firstFOStream, secondFOStream, mergedFOStream;
    InputStream[] input = InputStream[2];

    // creates XSL-FO from arguments
    firstFOStream = FOUtility.createFO(args[0], args[1]);

    // creates another XSL-FO from arguments
    secondFOStream = FOUtility.createFO(args[2], args[3]);

    // set each InputStream into the InputStream Array
    Array.set(input, 0, firstFOStream);
    Array.set(input, 1, secondFOStream);

    // merges two XSL-FOs
    mergedFOStream = FOUtility.mergeFOs(input);

    if (mergedFOStream == null)
    {
        System.out.println("Merge failed.");
        System.exit(1);
    }
    System.exit(0);
}

Merging Two XSL-FO Files
Input:
• Two XSL-FO file names (Array)
Output:
• One XSL-FO (InputStream)
Example

```java
import oracle.apps.xdo.template.fo.util.FOUtility;
.
.
public static void main(String[] args)
{
    InputStream mergedFOStream;

    // creates Array
    String[] input = {args[0], args[1]};

    // merges two FO files
    mergedFOStream = FOUtility.mergeFOs(input);
    if (mergedFOStream == null)
    {
        System.out.println("Merge failed.");
        System.exit(1);
    }
    System.exit(0);
}
```

Generating Output from an FO file

The FO Processor can also be used to process an FO object to generate your final output. An FO object is the result of the application of an XSL-FO stylesheet to XML data. These objects can be generated from a third party application and fed as input to the FO Processor.

The processor is called using a similar method to those already described, but a template is not required as the formatting instructions are contained in the FO.

Generating Output Using File Names

Input:
- FO file name (String)

Output:
- PDF file name (String)
Example
import oracle.apps.xdo.template.FOProcessor;
.
.
public static void main(String[] args) {

    FOProcessor processor = new FOProcessor();
    processor.setData(args[0]);    // set XSL-FO input file
    processor.setTemplate((String)null);
    processor.setOutput(args[2]);   // set (PDF) output file
    processor.setOutputFormat(FOProcessor.FORMAT_PDF);
    // Start processing
    try
    {
        processor.generate();
    }
    catch (XDOException e)
    {
        e.printStackTrace();
        System.exit(1);
    }

    System.exit(0);
}

Generating Output Using Streams

Input:
• FO data (InputStream)

Output:
• Output (OutputStream)
Example

```java
import java.io.InputStream;
import java.io.OutputStream;
import oracle.apps.xdo.template.FOProcessor;

public void runFOProcessor(InputStream xmlfoInputStream,
                             OutputStream pdfOutputStream)
{
    FOProcessor processor = new FOProcessor();
    processor.setData(xmlfoInputStream);
    processor.setTemplate((String)null);

    processor.setOutput(pdfOutputStream);
    // Set output format (for PDF generation)
    processor.setOutputFormat(FOProcessor.FORMAT_PDF);
    // Start processing
    try
    {
        processor.generate();
    }
    catch (XDOException e)
    {
        e.printStackTrace();
        System.exit(1);
    }
}
```

Generating Output with an Array of FO Data

Pass multiple FO inputs as an array to generate a single output file. A template is not required, therefore set the members of the template array to null, as shown in the example.

Input:
- FO data (Array)

Output:
- Output File Name (String)
Example
import java.lang.reflect.Array;
import oracle.apps.xdo.template.FOProcessor;
.
.
.
public static void main(String[] args)
{
    String[] xmlInput = {"first.fo", "second.fo", "third.fo");
    String[] xslInput = {null, null, null};  // null needs for xsl fo input

    FOProcessor processor = new FOProcessor();
    processor.setData(xmlInput);
    processor.setTemplate(xslInput);

    processor.setOutput("/tmp/output.pdf");  // set (PDF) output file
    processor.setOutputFormat(FOProcessor.FORMAT_PDF); processor.process();
    // Start processing
    try
    {
        processor.generate();
    }
    catch (XDOException e)
    {
        e.printStackTrace();
        System.exit(1);
    }
}

PDF Document Merger

The PDF Document Merger class provides a set of utilities to manipulate PDF documents. Using these utilities, you can merge documents, add page numbering, set backgrounds, and add watermarks.

Merging PDF Documents

Many business documents are composed of several individual documents that need to be merged into a single final document. The PDFDocMerger class supports the merging of multiple documents to create a single PDF document. This can then be manipulated further to add page numbering, watermarks, or other background images.

Merging with Input/Output File Names

The following code demonstrates how to merge (concatenate) two PDF documents using physical files to generate a single output document.

Input:

• PDF_1 file name (String)
• PDF_2 file name (String)

Output:
• PDF file name (String)

Example

```java
import java.io.*;
import oracle.apps.xdo.common.pdf.util.PDFDocMerger;

public static void main(String[] args) {
    try {
        // Last argument is PDF file name for output
        int inputNumbers = args.length - 1;

        // Initialize inputStreams
        FileInputStream[] inputStreams = new FileInputStream[inputNumbers];
        inputStreams[0] = new FileInputStream(args[0]);
        inputStreams[1] = new FileInputStream(args[1]);

        // Initialize outputStream
        FileOutputStream outputStream = new FileOutputStream(args[2]);

        // Initialize PDFDocMerger
        PDFDocMerger docMerger = new PDFDocMerger(inputStreams, outputStream);

        // Merge PDF Documents and generates new PDF Document
        docMerger.mergePDFDocs();
        docMerger = null;

        // Closes inputStreams and outputStream
    } catch (Exception exc) {
        exc.printStackTrace();
    }
}
```

Merging with Input/Output Streams

Input:
• PDF Documents (InputStream Array)

Output:
• PDF Document (OutputStream)
Example
import java.io.*;
import oracle.apps.xdo.common.pdf.util.PDFDocMerger;
.
.
public boolean mergeDocs(InputStream[] inputStreams, OutputStream outputStream)
{
    try {
        // Initialize PDFDocMerger
        PDFDocMerger docMerger = new PDFDocMerger(inputStreams, outputStream);

        // Merge PDF Documents and generates new PDF Document
        docMerger.mergePDFDocs();
        docMerger = null;

        return true;
    }
    catch(Exception exc)
    {
        exc.printStackTrace();
        return false;
    }
}

Merging with Background to Place Page Numbering
The following code demonstrates how to merge two PDF documents using input streams to generate a single merged output stream.

To add page numbers:
1. Create a background PDF template document that includes a PDF form field in the position that you would like the page number to appear on the final output PDF document.
2. Name the form field @pagenum@.
3. Enter the number in the field from which to start the page numbering. If you do not enter a value in the field, the start page number defaults to 1.

Input:
• PDF Documents (InputStream Array)
• Background PDF Document (InputStream)

Output:
• PDF Document (OutputStream)
Example
import java.io.*;
import oracle.apps.xdo.common.pdf.util.PDFDocMerger;
.
.
.
public static boolean mergeDocs(InputStream[] inputStreams, InputStream backgroundStream, OutputStream outputStream)
{
    try
    {
        // Initialize PDFDocMerger
        PDFDocMerger docMerger = new PDFDocMerger(inputStreams, outputStream);

        // Set Background
        docMerger.setBackground(backgroundStream);

        // Merge PDF Documents and generates new PDF Document
        docMerger.mergePDFDocs();
        docMerger = null;

        return true;
    }
    catch(Exception exc)
    {
        exc.printStackTrace();
        return false;
    }
}

Adding Page Numbers to Merged Documents

The FO Processor supports page numbering natively through the XSL-FO templates, but if you are merging multiple documents you must use this class to number the complete document from beginning to end.

The following code example places page numbers in a specific point on the page, formats the numbers, and sets the start value using the following methods:

- setPageNumberCoordinates (x, y) - sets the x and y coordinates for the page number position. The following example sets the coordinates to 300, 20.
- setPageNumberFontInfo (font name, size) - sets the font and size for the page number. If you do not call this method, the default "Helvetica", size 8 is used. The following example sets the font to "Courier", size 8.
- setPageNumberValue (n, n) - sets the start number and the page on which to begin numbering. If you do not call this method, the default values 1, 1 are used.

Input:
- PDF Documents (InputStream Array)
Output:
- PDF Document (OutputStream)
Example
import java.io.*;
import oracle.apps.xdo.common.pdf.util.PDFDocMerger;
.
.
public boolean mergeDocs(InputStream[] inputStreams, OutputStream outputStream)
{
    try {
        // Initialize PDFDocMerger
        PDFDocMerger docMerger = new PDFDocMerger(inputStreams, outputStream);

        // Calls several methods to specify Page Number
        // Calling setPageNumberCoordinates() method is necessary to set Page Numbering
        // Please refer to javadoc for more information
        docMerger.setPageNumberCoordinates(300, 20);

        // If this method is not called, then the default font "(Helvetica, 8)" is used.
        docMerger.setPageNumberFontInfo("Courier", 8);

        // If this method is not called, then the default initial value "(1, 1)" is used.
        docMerger.setPageNumberValue(1, 1);

        // Merge PDF Documents and generates new PDF Document
        docMerger.mergePDFDocs();
        docMerger = null;

        return true;
    } catch (Exception exc) {
        exc.printStackTrace();
        return false;
    }
}

Setting a Text or Image Watermark

Some documents that are in a draft phase require that a watermark indicating "DRAFT" be displayed throughout the document. Other documents might require a background image on the document. The following code sample shows how to use the PDFDocMerger class to set a watermark.

Setting a Text Watermark

Use the setTextDefaultWatermark() method to set a text watermark with the following attributes:
• Text angle (in degrees): 55
• Color: light gray (0.9, 0.9, 0.9)
• Font: Helvetica
• Font Size: 100
• The start position is calculated based on the length of the text

Alternatively, use the SetTextWatermark() method to set each attribute separately. Use the SetTextWatermark() method as follows:

• SetTextWatermark("Watermark Text", x, y) - declare the watermark text, and set the x and y coordinates of the start position. In the following example, the watermark text is "Draft" and the coordinates are 200f, 200f.

• setTextWatermarkAngle (n) - sets the angle of the watermark text. If this method is not called, 0 will be used.

• setTextWatermarkColor (R, G, B) - sets the RGB color. If this method is not called, light gray (0.9, 0.9, 0.9) will be used.

• setTextWatermarkFont ("font name", font size) - sets the font and size. If you do not call this method, Helvetica, 100 will be used.

The following example shows how to set these properties and then call the PDFDocMerger.

Input:
• PDF Documents (InputStream)

Output:
• PDF Document (OutputStream)
Example

```java
import java.io.*;
import oracle.apps.xdo.common.pdf.util.PDFDocMerger;
...
public boolean mergeDocs(InputStream inputStreams, OutputStream outputStream)
{
    try
    {
        // Initialize PDFDocMerger
        PDFDocMerger docMerger = new PDFDocMerger(inputStreams, outputStream);

        // You can use setTextDefaultWatermark() without these detailed setting
        docMerger.setTextWatermark("DRAFT", 200f, 200f); // set text and place
        docMerger.setTextWatermarkAngle(80); // set angle
        docMerger.setTextWatermarkColor(1.0f, 0.3f, 0.5f); // set RGB Color

        // Merge PDF Documents and generates new PDF Document
        docMerger.mergePDFDocs();
        docMerger = null;

        return true;
    }
    catch(Exception exc)
    {
        exc.printStackTrace();
        return false;
    }
}
```

Setting Image Watermark

An image watermark can be set to cover the entire background of a document, or just to cover a specific area (for example, to display a logo). Specify the placement and size of the image using rectangular coordinates as follows:

```java
float[ ] rct = {LowerLeft X, LowerLeft Y, UpperRight X, UpperRight Y}
```

For example:

```java
float[ ] rct = {100f, 100f, 200f, 200f}
```

The image will be sized to fit the rectangular area defined.

To use the actual image size, without sizing it, define the LowerLeft X and LowerLeft Y positions to define the placement and specify the UpperRight X and UpperRight Y coordinates as -1f. For example:

```java
float[ ] rct = {100f, 100f, -1f, -1f}
```
Input:
- PDF Documents (InputStream)
- Image File (InputStream)

Output:
- PDF Document (OutputStream)

Example
```java
import java.io.*;
import oracle.apps.xdo.common.pdf.util.PDFDocMerger;

public boolean mergeDocs(InputStream inputStreams, OutputStream outputStream, String imageFilePath)
{
    try {
        // Initialize PDFDocMerger
        PDFDocMerger docMerger = new PDFDocMerger(inputStreams, outputStream);

        FileInputStream wmStream = new FileInputStream(imageFilePath);
        float[] rct = {100f, 100f, -1f, -1f};
        pdfMerger.setImageWatermark(wmStream, rct);

        // Merge PDF Documents and generates new PDF Document
        docMerger.mergePDFDocs();
        docMerger = null;

        // Closes inputStreams
        return true;
    } catch (Exception exc) {
        exc.printStackTrace();
        return false;
    }
}
```

**PDF Book Binder Processor**

The PDFBookBinder processor is useful for the merging of multiple PDF documents into a single document consisting of a hierarchy of chapters, sections, and subsections and a table of contents for the document. The processor also generates PDF style “bookmarks”; the outline structure is determined by the chapter and section hierarchy. The processor is extremely powerful allowing you complete control over the combined document.

This section provides only an overview of the features and usage of the PDF Book Binder Processor. For detailed information, please see OracleMetaLink note 358886.1, "Oracle XML Publisher PDF Binding Examples."
Usage

The table of contents formatting and style is created through the use of an RTF template created in Microsoft Word. The chapters are passed into the program as separate PDF files (one chapter, section, or subsection corresponds to one PDF file). Templates may also be specified at the chapter level for insertion of dynamic or static content, page numbering, and placement of hyperlinks within the document.

The templates can be in RTF or PDF format. RTF templates are more flexible by allowing you to leverage XML Publisher’s support for dynamic content. PDF templates are much less flexible, making it difficult to achieve desirable effects such as the reflow of text areas when inserting page numbers and other types of dynamic content.

The templates can be rotated (at right angles) or be made transparent. A PDF template can also be specified at the book level, enabling the ability to specify global page numbering, or other content such as backgrounds and watermarks. A title page can also be passed in as a parameter, as well as cover and closing pages for each chapter or section.

XML Control File

The structure of the book’s chapters, sections, and subsections is represented as XML and passed in as a command line parameter; or it can also be passed in at the API level. All of the chapter and section files, as well as all the templates files and their respective parameters, are specified inside this XML structure. Therefore, the only two required parameters are an XML file and a PDF output file.

You can also specify volume breaks inside the book structure. Specifying volume breaks will split the content up into separate output files for easier file and printer management.

The structure of the XML control file is represented in the following diagram:
To specify template and content file locations in your XML structure, you can specify a path relative to your local file system or you can specify a URL referring to the template or content location. Secure HTTP protocol is supported, as well as specially recognized XML Publisher protocols, such as:

- “xdo://” - used to specify XML Publisher Template Manager-specific data.
- “fnd://” - used to specify data located in the FND_LOBS table.
- “blob://” - used for specifying data in any user-defined BLOB table.

The format for the “blob://” protocol is:

```
blob://[table_name].[blob_column_name]/[pk_datatype]:[pk_name]=[pk_value]/../../..
```

**Command Line Options**

Following is an example of the command line usage:

```
```

where
-xml <file> is the file name of the input XML file containing the table of contents XML structure.

-pdf <file> is the final generated PDF output file.

-temp <directory> is the temporary directory for better memory management. (This is optional, if not specified, the system environment variable “java.io.tmpdir” will be used.)

-log <file> sets the output log file (optional, default is System.out).

-debug <true or false> turns debugging off or on.

**API Method Call**

The following is an example of an API method call:

```java
String xmlInputPath = "c:\tmp\toc.xml";
String pdfOutputPath = "c:\tmp\final_book.pdf";
PDFBookBinder bookBinder = new PDFBookBinder(xmlInputPath,
                                              pdfOutputPath);

bookBinder.setConfig(new Properties());
bookBinder.process();
```

**Document Processor Engine**

The Document Processor Engine provides batch processing functionality to access a single API or multiple APIs by passing a single XML instance document to specify template names, data sources, languages, output type, output names, and destinations.

This solution enables batch printing with XML Publisher, in which a single XML document can be used to define a set of invoices for customers, including the preferred output format and delivery channel for those customers. The XML format is very flexible allowing multiple documents to be created or a single master document.

This section:

- Describes the hierarchy and elements of the Document Processor XML file
- Provides sample XML files to demonstrate specific processing options
- Provides example code to invoke the processors

**Hierarchy and Elements of the Document Processor XML File**

The Document Processor XML file has the following element hierarchy:
Requestset
request
delivery
  filesystem
  print
  fax
  number
  email
  message
document
  background
text
pagenumber
template
data

This hierarchy is displayed in the following illustration:

```
requestset
  request
    delivery
      filesystem
      email
      fax
      print
      message
      number
    document
      template
      pagenumber
      background
      data
      text
```

The following table describes each of the elements:

<table>
<thead>
<tr>
<th>Element</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestset</td>
<td>xmlns, version</td>
<td>Root element must contain [xmlns:xapi=&quot;<a href="http://xmlns.oracle.com/oxp/xapi/">http://xmlns.oracle.com/oxp/xapi/</a>&quot;] block. The version is not required, but defaults to &quot;1.0&quot;.</td>
</tr>
<tr>
<td>request</td>
<td>N/A</td>
<td>Element that contains the data and template processing definitions.</td>
</tr>
<tr>
<td>delivery</td>
<td>N/A</td>
<td>Defines where the generated output is sent.</td>
</tr>
<tr>
<td>Element</td>
<td>Attributes</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>document</td>
<td>output-type</td>
<td>Specify one output that can have several template elements. The output-type attribute is optional. Valid values are: pdf (Default) rtf html excel text</td>
</tr>
<tr>
<td>filesystem</td>
<td>output</td>
<td>Specify this element to save the output to the file system. Define the directory path in the output attribute.</td>
</tr>
<tr>
<td>print</td>
<td>• printer • server-alias</td>
<td>The print element can occur multiple times under delivery to print one document to several printers. Specify the printer attribute as a URI, such as: &quot;ipp:/ /myprintserver:631/ printers/printername&quot;</td>
</tr>
<tr>
<td>fax</td>
<td>• server • server-alias</td>
<td>Specify a URI in the server attribute, for example: &quot;ipp:/ //myfaxserver1:631/ printers/myfaxmachine&quot;</td>
</tr>
<tr>
<td>number</td>
<td></td>
<td>The number element can occur multiple times to list multiple fax numbers. Each element occurrence must contain only one number.</td>
</tr>
<tr>
<td>email</td>
<td>• server • port • from • reply-to • server-alias</td>
<td>Specify the outgoing mail server (SMTP) in the server attribute. Specify the mail server port in the port attribute.</td>
</tr>
<tr>
<td>Element</td>
<td>Attributes</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>message</td>
<td>• to</td>
<td>The <code>message</code> element can be placed several times under the <code>email</code> element. You can specify character data in the <code>message</code> element.</td>
</tr>
<tr>
<td></td>
<td>• cc</td>
<td>You can specify multiple e-mail addresses in the <code>to</code>, <code>cc</code> and <code>bcc</code> attributes separated by a comma.</td>
</tr>
<tr>
<td></td>
<td>• bcc</td>
<td>The <code>attachment</code> value is either true or false (default). If <code>attachment</code> is true, then a generated document will be attached when the e-mail is sent.</td>
</tr>
<tr>
<td></td>
<td>• attachment</td>
<td>The <code>subject</code> attribute is optional.</td>
</tr>
<tr>
<td></td>
<td>• subject</td>
<td>background</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If the background text is required on a specific page, then set the <code>where</code> value to the page numbers required. The page index starts at 1. The default value is 0, which places the background on all pages.</td>
</tr>
<tr>
<td>text</td>
<td>• title</td>
<td>Specify the watermark text in the <code>title</code> value.</td>
</tr>
<tr>
<td></td>
<td>• default</td>
<td>A default value of &quot;yes&quot; automatically draws the watermark with forward slash type. The default value is yes.</td>
</tr>
<tr>
<td>pagenumber</td>
<td>• initial-page-index</td>
<td>The <code>initial-page-index</code> default value is 0.</td>
</tr>
<tr>
<td></td>
<td>• initial-value</td>
<td>The <code>initial-value</code> default value is 1.</td>
</tr>
<tr>
<td></td>
<td>• x-pos</td>
<td>&quot;Helvetica&quot; is used for the page number font.</td>
</tr>
<tr>
<td></td>
<td>• y-pos</td>
<td>The <code>x-pos</code> provides lower left x position.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The <code>y-pos</code> provides lower left y position.</td>
</tr>
<tr>
<td>Element</td>
<td>Attributes</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| template | • locale  
• location  
• type | Contains template information. Valid values for the type attribute are pdf rtf xsl-fo etext The default value is "pdf". |
| data     | location   | Define the location attribute to specify the location of the data, or attach the actual XML data with subelements. The default value of location is "inline". If the location points to either an XML file or a URL, then the data should contain an XML declaration with the proper encoding. If the location attribute is not specified, the data element should contain the subelements for the actual data. This must not include an XML declaration. |

**XML File Samples**

Following are sample XML files that show:

- Simple XML shape
- Defining two data sets
- Defining multiple templates and data
- Retrieving templates over HTTP
- Retrieving data over HTTP
- Generating more than one output
- Defining page numbers

**Simple XML sample**

The following sample is a simple example that shows the definition of one template (template1.pdf) and one data source (data1) to produce one output file (outfile.pdf) delivered to the file system:
Example
<?xml version="1.0" encoding="UTF-8" ?>
<xapi:requestset xmlns:xapi="http://xmlns.oracle.com/oxp/xapi">
  <xapi:request>
    <xapi:delivery>
      <xapi:filesystem output="d:\tmp\outfile.pdf"/>
    </xapi:delivery>
    <xapi:document output-type="pdf">
      <xapi:template type="pdf" location="d:\mywork\template1.pdf">
        <xapi:data>
          <field1>The first set of data</field1>
        </xapi:data>
        <xapi:data>
          <field1>The second set of data</field1>
        </xapi:data>
      </xapi:template>
    </xapi:document>
  </xapi:request>
</xapi:requestset>

Defining two data sets

The following example shows how to define two data sources to merge with one template to produce one output file delivered to the file system:

Example
<?xml version="1.0" encoding="UTF-8" ?>
<xapi:requestset xmlns:xapi="http://xmlns.oracle.com/oxp/xapi">
  <xapi:request>
    <xapi:delivery>
      <xapi:filesystem output="d:\tmp\outfile.pdf"/>
    </xapi:delivery>
    <xapi:document output-type="pdf">
      <xapi:template type="pdf" location="d:\mywork\template1.pdf">
        <xapi:data>
          <field1>The first set of data</field1>
        </xapi:data>
        <xapi:data>
          <field1>The second set of data</field1>
        </xapi:data>
      </xapi:template>
    </xapi:document>
  </xapi:request>
</xapi:requestset>

Defining multiple templates and data

The following example builds on the previous examples by applying two data sources to one template and two data sources to a second template, and then merging the two into a single output file. Note that when merging documents, the output-type must be "pdf".
Example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xapi:requestset xmlns:xapi="http://xmlns.oracle.com/oxp/xapi">
  <xapi:request>
    <xapi:delivery>
      <xapi:filesystem output="d:\tmp\outfile3.pdf"/>
    </xapi:delivery>

    <xapi:document output-type="pdf">
      <xapi:template type="pdf" location="d:\mywork\template1.pdf">
        <xapi:data>
          <field1>The first set of data</field1>
        </xapi:data>
        <xapi:data>
          <field1>The second set of data</field1>
        </xapi:data>
      </xapi:template>

      <xapi:template type="pdf" location="d:\mywork\template2.pdf">
        <xapi:data>
          <field1>The third set of data</field1>
        </xapi:data>
        <xapi:data>
          <field1>The fourth set of data</field1>
        </xapi:data>
      </xapi:template>
    </xapi:document>
  </xapi:request>
</xapi:requestset>
```

Retrieving templates over HTTP

This sample is identical to the previous example, except in this case the two templates are retrieved over HTTP:
Retrieving data over HTTP

This sample builds on the previous example and shows one template with two data sources, all retrieved via HTTP; and a second template retrieved via HTTP with its two data sources embedded in the XML:
Example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xapi:requestset xmlns:xapi="http://xmlns.oracle.com/oxp/xapi">
  <xapi:request>
    <xapi:delivery>
      <xapi:filesystem output="d:\temp\out5.pdf"/>
    </xapi:delivery>

    <xapi:document output-type="pdf">
      <xapi:template type="pdf" location="http://your.server:9999/templates/template1.pdf">
        <xapi:data location="http://your.server:9999/data/data_1.xml"/>
        <xapi:data location="http://your.server:9999/data/data_2.xml"/>
      </xapi:template>

      <xapi:template type="pdf" location="http://your.server:9999/templates/template2.pdf">
        <xapi:data>
          <field1>The third page data</field1>
        </xapi:data>
        <xapi:data>
          <field1>The fourth page data</field1>
        </xapi:data>
      </xapi:template>
    </xapi:document>
  </xapi:request>
</xapi:requestset>
```

Generating more than one output

The following sample shows the generation of two outputs: `out_1.pdf` and `out_2.pdf`. Note that a request element is defined for each output.
Example
<?xml version="1.0" encoding="UTF-8"?>
<xapi:requestset xmlns:xapi="http://xmlns.oracle.com/oxp/xapi">
  <xapi:request>
    <xapi:delivery>
      <xapi:filesystem output="d:\temp\out_1.pdf"/>
    </xapi:delivery>
    <xapi:document output-type="pdf">
      <xapi:template type="pdf"
        location="d:\mywork\template1.pdf">
        <xapi:data>
          <field1>The first set of data</field1>
        </xapi:data>
        <xapi:data>
          <field1>The second set of data</field1>
        </xapi:data>
      </xapi:template>
    </xapi:document>
  </xapi:request>
  <xapi:request>
    <xapi:delivery>
      <xapi:filesystem output="d:\temp\out_2.pdf"/>
    </xapi:delivery>
    <xapi:document output-type="pdf">
      <xapi:template type="pdf"
        location="d:\mywork\template2.pdf">
        <xapi:data>
          <field1>The third set of data</field1>
        </xapi:data>
        <xapi:data>
          <field1>The fourth set of data</field1>
        </xapi:data>
      </xapi:template>
    </xapi:document>
  </xapi:request>
</xapi:requestset>

Defining page numbers

The following sample shows the use of the pagenumber element to define page numbers on a PDF output document. The first document that is generated will begin with an initial page number value of 1. The second output document will begin with an initial page number value of 3. The pagenumber element can reside anywhere within the document element tags.

Note that page numbering that is applied using the pagenumber element will not replace page numbers that are defined in the template.
<xml version="1.0" encoding="UTF-8"/>
<xapi:requestset xmlns:xapi="http://xmlns.oracle.com/oxp/xapi">
  <xapi:request>
    <xapi:delivery>
      <xapi:filesystem output="d:\temp\out7-1.pdf"/>
    </xapi:delivery>
    <xapi:document output-type="pdf">
      <xapi:template type="pdf" location="d:\mywork\template1.pdf">
        <xapi:data>
          <field1>The first page data</field1>
        </xapi:data>
        <xapi:data>
          <field1>The second page data</field1>
        </xapi:data>
      </xapi:template>
      <xapi:pagenumber initial-value="1" initial-page-index="1" x-pos="300" y-pos="20"/>
    </xapi:document>
  </xapi:request>

  <xapi:request>
    <xapi:delivery>
      <xapi:filesystem output="d:\temp\out7-2.pdf"/>
    </xapi:delivery>
    <xapi:document output-type="pdf">
      <xapi:template type="pdf" location="d:\mywork\template2.pdf">
        <xapi:data>
          <field1>The third page data</field1>
        </xapi:data>
        <xapi:data>
          <field1>The fourth page data</field1>
        </xapi:data>
      </xapi:template>
      <xapi:pagenumber initial-value="3" initial-page-index="1" x-pos="300" y-pos="20"/>
    </xapi:document>
  </xapi:request>
</xapi:requestset>

Invoke Processors

The following code samples show how to invoke the document processor engine using an input file name and an input stream.

Invoke Processors with Input File Name

Input:

- Data file name (String)
- Directory for Temporary Files (String)
Example
import oracle.apps.xdo.batch.DocumentProcessor;
.
.
.
public static void main(String[] args)
{
.
.
.
try
{
    // dataFile --- File path of the Document Processor XML
    // tempDir --- Temporary Directory path
    DocumentProcessor docProcessor = new DocumentProcessor(data
    File, tempDir);
    docProcessor.process();
}
    catch(Exception e)
    {
    e.printStackTrace();
    System.exit(1);
    }
    System.exit(0);
}

Invoke Processors with InputStream
Input:
• Data file (InputStream)
• Directory for Temporary Files (String)
Example

```java
import oracle.apps.xdo.batch.DocumentProcessor;
import java.io.InputStream;

public static void main(String[] args)
{
    try
    {
        // dataFile --- File path of the Document Processor XML
        // tempDir --- Temporary Directory path
        FileInputStream fIs = new FileInputStream(dataFile);

        DocumentProcessor docProcessor = new DocumentProcessor(fIs, tempDir);
        docProcessor.process();
        fIs.close();
    }
    catch(Exception e)
    {
        e.printStackTrace();
        System.exit(1);
    }
    System.exit(0);
}
```

Bursting Engine

XML Publisher’s bursting engine accepts a data stream and splits it based on multiple criteria, generates output based on a template, then delivers the individual documents through the delivery channel of choice. The engine provides a flexible range of possibilities for document generation and delivery. Example implementations include:

- Invoice generation and delivery based on customer-specific layouts and delivery preference
- Financial reporting to generate a master report of all cost centers, bursting out individual cost center reports to the appropriate manager
- Generation of payslips to all employees based on one extract and delivered via e-mail

Usage

The bursting engine is an extension of the Document Processor Engine, page 10-31 and has its own method be called to invoke it. The Document Processor CML structure has been extended to handle the new components required by the bursting engine. It supports all of the delivery functionality that the Document Processor supports using the same format. It accepts the XML data to be burst and a control file that takes the Document Processor XML format (see Hierarchy and Elements of the Document Processor XML File, page 10-31).
Control File

The control file takes the same format as the Document Processor XML, page 10-31 with a few extensions:

- Use the attribute select under the request element to specify the element in the XML data that you wish to burst on.

  **Example**
  
  <xapi:request select="/EMPLOYEES/EMPLOYEE"/>

- Use the attribute id under the lowest level of the delivery structure (for example, for the delivery element email, the id attribute belongs to the message element. This assigns an ID to the delivery method to be referenced later in the XML file.

  **Example**
  
  <xapi:message id="123" to="jo.smith@company.com"/>

- Use the delivery attribute under the document element. This assigns the delivery method for the generated document as defined in the id attribute for the delivery element. You can specify multiple delivery channels separated by a comma.

  **Example**
  
  <xapi:document output-type="pdf" delivery="123"/>

- Use the filter attribute on the template element. Use this to apply a layout template based on a filter on your XML data.

  **Example**
  
  <xapi:template type="rtf" location="/usr/tmp/empGeneric.rtf"/>
  
  <xapi:template type="rtf" location="/usr/tmp/empDet.rtf" filter="./EMPLOYEE[ENAME='SMITH']"/>

  This will apply the empDet template only to those employees with the name "SMITH". All other employees will have the empGeneric template applied. This filter can use any XPATH expression to determine the rules for the template application.

Dynamic Delivery Destination

You can reference elements in the data to derive certain delivery attributes, such as an e-mail address or fax number. Enter the value for the attribute using the following form:

\${ELEMENT}

where ELEMENT is the element name from the XML data that holds the value for the attribute.

For example:

<xapi:message id="123" to="\${EMAIL}"/>

At runtime the value of the to attribute will be set to the value of the EMAIL element from the input XML file.

You can also set the value of an attribute by passing a parameter to API in a Properties object.

Dynamic Delivery Content

You can reference information in the XML data to be put into the delivery content. This takes the same format described above (that is, \${ELEMENT}).
For example, suppose you wanted to burst a document to employees via e-mail and personalize the e-mail by using the employee’s name in the subject line. Assuming the employee’s name is held in an element called ENAME, you could use ${ENAME} to reference the employee’s name in the control file as follows:

subject="Employee Details for ${ENAME}"

**Sample Control File**

The following sample control file shows an example control file to split data based on an EMPLOYEE element and send an e-mail to each employee with their own data. The sample file is annotated.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xapi:requestset xmlns:xapi="http://xmlns.oracle.com/oxp/xapi">
  <xapi:request select="/EMPLOYEES/EMPLOYEE">
    <! - This sets the bursting element i.e., EMPLOYEE - >
    <xapi:delivery>
      <xapi:email server="rgnamerssmtp.oraclecorp.com" port="25"
        from="xmlpadmin1@oracle.com" reply-to = "reply@oracle.com">
        <xapi:message id="123" to="${EMAIL}" cc="${EMAIL_ALL}"
          attachment="true" subject="Employee Details for ${ENAME}"
          > Mr. ${ENAME}, Please review the attached document</xapi:message>
      </xapi:email>
      <! - This assigns a delivery id of '123'. It also sets the e-mail address of the employee and a cc copy to a parameter value EMAIL_ALL; this might be a manager’s e-mail. The employee’s name (ENAME) can also be used in the subject/body of the email. - >
      </xapi:delivery>
    </xapi:request>
  </xapi:requestset>
</xapi:requestset>
```

**Multiple Bursting Options**

The bursting engine can support multiple bursting criteria and delivery options. Assume you have a report that generates data for all employees with their manager’s information. You can construct a control file that will:

- Burst the employee data to each employee
- Burst a report to each manager that contains the data about his employees

You can provide a different template for each bursting level. You can therefore generate the employee report based on one template and the summary manager’s report based on a different template, but still use the same data set.

To achieve this multibursting result, you must add a second request element to the control file structure.
Multibursting Example

The following sample shows how to construct a control file that will burst on the 
EMPLOYEE level and the MANAGER level:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<xapi:requestset xmlns:xapi="http://xmlns.oracle.com/oxp/xapi">
  <! - First request to burst on employee ->
  <xapi:request select="/EMPLOYEES/EMPLOYEE">
    <xapi:delivery>
      <xapi:email /><server details removed>>
      <xapi:message id="123" /><message details removed>>
      </xapi:message>
    </xapi:email>
    <xapi:fax server="ipp://mycupsserver:631/printers/fax2">
      <xapi:number id="FAX1">916505069560</xapi:number>
    </xapi:fax>
    <xapi:print id="printer1" 
      printer="ipp://mycupsserver:631/printers/printer1" 
      copies="2" />
    </xapi:delivery>
    <xapi:document output-type="pdf" delivery="123">
      <xapi:template type="rtf" location="usr\tmp\empDet.rtf" />
    </xapi:document>
  </xapi:request>
  <!Second request to burst on department ->
  <xapi:request select="/DATA/DEPT/MANAGER">
    <xapi:delivery>
      <xapi:email server="gmsmtp.oraclecorp.com" port="" 
        from="XDOburstingTest@oracle.com" reply-to="reply@oracle.com">
      <xapi:message id="123" to="${MANAGER_EMAIL}" 
        cc="${MANAGER_EMAIL}" attachment="true" 
        subject="Department Summary for ${DEPTNO}>Please review 
        the attached Department Summary for department ${DEPTNO}</xapi:message>
      </xapi:email>
    </xapi:delivery>
    <xapi:document output-type="rtf" delivery="123">
      <xapi:template type="rtf" 
        location="d:\burst_test\deptSummary.rtf" />
    </xapi:document>
  </xapi:request>
</xapi:requestset>
```

Bursting Listeners

The bursting engine provides a listening interface that allows you to listen to the 
various stages of the bursting process. Following are the supported modes that you 
can subscribe to:

- `beforeProcess()` - before the bursting process starts.
- `afterProcess()` - after the bursting process completes.
- `beforeProcessRequest(int requestIndex)` - before the bursting request 
  starts. This interface provides an assigned request ID for the current request.
- `afterProcessRequest(int requestIndex)` - after the bursting request has 
  completed; provides the request ID for the current request.
• **beforeProcessDocument**(int requestIndex, int documentIndex, String deliveryId) - before the document generation starts; provides the request ID and a document ID.

• **afterProcessDocument**(int requestIndex, int documentIndex, Vector documentOutputs) - after the document generation completes; provides the request ID and document ID, plus a Vector list of the document objects generated in the request.

• **beforeDocumentDelivery**(int requestIndex, int documentIndex, String deliveryId) - before the documents in the request are delivered; provides the request ID, the document ID, and a delivery ID.

• **afterDocumentDelivery**(int requestIndex, int documentIndex, String deliveryId, Object deliveryObject, Vector attachments) - after the document delivery completes; provides a request ID, document ID, and delivery ID, plus a Vector list of the documents delivered in the request.

You can subscribe to any of these interfaces in your calling Java class. The listeners are useful to determine if the processing of individual documents is proceeding successfully or to start another process based on the successful completion of a request.

**Calling the Bursting API**

To call the bursting API, instantiate an instance of DocumentProcessor class using one of the following formats:

```
DocumentProcessor(xmlCtrlInput, xmlDataInput, tmpDir)
where
```

xmlCtrlInput - is the control file for the bursting process. This can be a string reference to a file, an inputStream object, or a Reader object.

xmlDataInput - is the XML data to be burst. This can a string reference to a file, an inputStream object, or a Reader object.

tmpDir - is a temporary working directory. This takes the format of a String object. This is optional as long as the main XML Publisher temporary directory has been set.

**Simple Example Java Class**

The following is a sample Java class:
public class BurstingTest {
    public BurstingTest() {
        try {
            DocumentProcessor dp = new DocumentProcessor("\burst\burstCtrl.xml", "\burst\empData.xml","\burst");
            dp.process();
        }
        catch (Exception e) {
            System.out.println(e);  
        }
    public static void main(String[] args) {
        BurstingTest burst1 = new BurstingTest();
    }
}

Example Java Class with Listeners
To take advantage of the bursting listeners, add the interface to the class declaration and use the registerListener method. Then code for the listeners you want to subscribe to as follows:

public class BurstingTest implements BurstingListener {
    public BurstingTest() {
        try {
            DocumentProcessor dp = new DocumentProcessor("\burst\burstCtrl.xml", "\burst\empData.xml","\burst");
            dp.registerListener(this);
            dp.process();
        }
        catch (Exception e) {
            System.out.println(e);  
        }
    public static void main(String[] args) {
        BurstingTest burst1 = new BurstingTest();
    }

    public void beforeProcess() {
        System.out.println("Start of Bursting Process");
    }

    public void afterProcess() {
        System.out.println("End of Bursting Process");
    }

    public void beforeProcessRequest(int requestIndex) {
        System.out.println("Start of Process Request ID"+requestIndex);
    }
}
public void afterProcessRequest(int requestIndex) {
    System.out.println("End of Process Request ID" + requestIndex);
}

public void beforeProcessDocument(int requestIndex, int documentIndex) {
    System.out.println("Start of Process Document");
    System.out.println(" Request Index "+requestIndex);
    System.out.println(" Document Index "+documentIndex);
}

public void afterProcessDocument(int requestIndex, int documentIndex, Vector documentOutputs) {
    System.out.println(" ========End of Process Document");
    System.out.println(" Outputs ":"+documentOutputs);
}

public void beforeDocumentDelivery(int requestIndex, int documentIndex, String deliveryId) {
    System.out.println(" ========Start of Delivery");
    System.out.println(" Request Index "+requestIndex);
    System.out.println(" Document Index "+documentIndex);
    System.out.println(" DeliveryId "+deliveryId);
}

public void afterDocumentDelivery(int requestIndex, int documentIndex, String deliveryId, Object deliveryObject, Vector attachments) {
    System.out.println(" ========End of Delivery");
    System.out.println(" Attachments ":"+attachments);
}

Passing a Parameter
To pass a parameter holding a value to be used in the control file for delivery, add the following code:

... Properties prop = new Properties();
    prop.put("user-variable:ADMIN_EMAIL","jo.smith@company.com");
    dp.setConfig(prop);
    dp.process();
...

Bursting Control File Examples
All of the examples in this section use the following XML data source:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Data>
```
Example 1 - Bursting Employee Data to Employees via E-mail
The following sample shows how to apply a template (empDet.rtf) to every employee's data, generate a PDF document, and deliver the document to each employee via e-mail.

<?xml version="1.0" encoding="UTF-8"?>
<xapi:requestset xmlns:xapi="http://xmlns.oracle.com/oxp/xapi">
  <xapi:request select="/DATA/DEPTS/DEPT/EMPLOYEES/EMPLOYEE">
    <! - Burst on employee element - >
    <xapi:delivery>
      <xapi:email server="my.smtp.server" port="25" from="xmlpadmin@mycomp.com" reply-to="">
        <xapi:message id="123" to="${EMAIL}"
          <! - Set the id for the delivery method - >
          <! - Use the employees EMAIL element to email the document to the employee - >
          cc="${ADMIN_EMAIL}"
          <! - Use the ADMIN_EMAIL parameter to CC the document to the administrator - >
          attachment="true" subject="Employee Details for ${ENAME}"/>
        Mr. ${ENAME}, Please review the attached document</xapi:message>
      </xapi:email>
    </xapi:delivery>
    <! - Embed the employees name into the email message - >
    <! - Specify the delivery method id to be used - >
    </xapi:delivery>
    <xapi:template type="rtf" location="/usr\empDet.rtf"></xapi:template>
  </xapi:request>
</xapi:requestset>

Example 2 - Bursting Employee Data to Employees via Multiple Delivery Channels and Conditionally Using Layout Templates
This sample shows how to burst, check the employee name, and generate a PDF using the appropriate template. The documents will then be e-mailed and printed.
Example 3 - Bursting Employee Data to Employees and Their Manager
This sample shows how to burst an e-mail with a PDF attachment to all employees using the empDet template. It will also burst an employee summary PDF to the manager of each department via e-mail.
XML Publisher Properties

The FO Processor supports PDF security and other properties that can be applied to your final documents. Security properties include making a document unprintable and applying password security to an encrypted document.

Other properties allow you to define font subsetting and embedding. If your template uses a font that would not normally be available to XML Publisher at runtime, you can use the font properties to specify the location of the font. At runtime XML Publisher will retrieve and use the font in the final document. For example, this property might be used for check printing for which a MICR font is used to generate the account and routing numbers on the checks.

See XML Publisher Properties, page 8-2 for the full list of properties.

Setting Properties

The properties can be set in two ways:

• At runtime, specify the property as a Java Property object to pass to the FO Processor.
• Set the property in a configuration file.
• Set the property in the template (RTF templates only). See Setting Properties, page 2-81 in the RTF template for this method.

Passing Properties to the FO Engine
To pass a property as a Property object, set the name/value pair for the property prior to calling the FO Processor, as shown in the following example:

Input:
• XML file name (String)
• XSL file name (String)

Output:
• PDF file name (String)

Example
import oracle.apps.xdo.template.FOProcessor;
.
.
public static void main(String[] args) {
    FOProcessor processor = new FOProcessor();
    processor.setData(args[0]); // set XML input file
    processor.setTemplate(args[1]); // set XSL input file
    processor.setOutput(args[2]); // set (PDF) output file
    processor.setOutputFormat(FOProcessor.FORMAT_PDF);
    Properties prop = new Properties();
    prop.put("pdf-security", "true"); // PDF security control
    prop.put("pdf-permissions-password", "abc"); // permissions password
    prop.put("pdf-encryption-level", "0"); // encryption level
    processor.setConfig(prop); // Start processing
    try {
        processor.generate();
    } catch (XDOException e) {
        e.printStackTrace();
        System.exit(1);
    }
    System.exit(0);
}

Passing a Configuration File to the FO Processor

The following code shows an example of passing the location of a configuration file.

Input:
• XML file name (String)
• XSL file name (String)

Output:
• PDF file name (String)

```java
import oracle.apps.xdo.template.FOProcessor;

public static void main(String[] args)
{
    FOProcessor processor = new FOProcessor();
    processor.setData(args[0]);  // set XML input file
    processor.setTemplate(args[1]);  // set XSL input file
    processor.setOutput(args[2]);  // set (PDF) output file
    processor.setOutputFormat(FOProcessor.FORMAT_PDF);
    processor.setConfig("/tmp/xmlpconfig.xml");
    // Start processing
    try
    {
        processor.generate();
    }
    catch (XDOException e)
    {
        e.printStackTrace();
        System.exit(1);
    }
    System.exit(0);
}
```

**Passing Properties to the Document Processor**

Input:
• Data file name (String)
• Directory for Temporary Files (String)

Output:
• PDF File
Example

```java
import oracle.apps.xdo.batch.DocumentProcessor;

public static void main(String[] args)
{
    try
    {
        // dataFile --- File path of the Document Processor XML
        // tempDir --- Temporary Directory path
        DocumentProcessor docProcessor = new DocumentProcessor(dataFile, tempDir);
        Properties prop = new Properties();
        prop.put("pdf-security", "true"); // PDF security control
        prop.put("pdf-permissions-password", "abc"); // permissions password
        prop.put("pdf-encryption-level", "0"); // encryption level
        processor.setConfig(prop);
        docProcessor.process();
    } catch (Exception e)
    {
        e.printStackTrace();
        System.exit(1);
    }
    System.exit(0);
}
```

Applications Layer APIs

The applications layer of XML Publisher allows you to store and manager data sources and templates through the Template Manager user interface via the XML Publisher Administrator responsibility. You can also access and manipulate these objects via an application program interfaces. This section describes the APIs that are available to a programmer.

Data sources and templates are stored in the database. This includes the metadata describing the object and the physical object itself (for example, an RTF file). Use these APIs to register, update, and retrieve information about datasources and templates. You can also call use the APIs to call XML Publisher to apply a template to a data source to generate output documents directly (without going through the concurrent manager).

In the XML Publisher schema, each data source can have multiple templates assigned to it. However, templates cannot exist without a data source. The following graphic illustrates this relationship:
Datasource APIs

The following APIs are provided to access and manipulate the data definitions programmatically:

- DataSource Class
- DataSourceHelper Class

DataSource Class

The data source acts as a placeholder object against which you register templates. The DataSource class represents a single data source definition entry. This class provides the following methods to set and get properties from the data source:

DataSourceHelper Class

This is a utility class that can be used to manage data source definition entries in the Template Manager repository.

A data source definition entry is a logical unit that represents a data source for the template. Each data source definition entry can have one data definition in XSD (XML Schema Definition) format, and one sample data file in XML. Each data source definition entry can have one or more display names and descriptions for each language. The appropriate name and description will be picked up and shown to users based on the user’s session language.

Getting AppsContext

All methods require the AppsContext instance to communicate with the Applications database. Use one of the following methods to get the AppsContext instance in your code.

1. If you are using this class in OA Framework, obtain AppsContext by calling

   ```java
   ((OADBTransactionImpl)am.getOADBTransaction()).getAppsContext()
   ```

   where `am` is your OA ApplicationModule.

2. If you are using this class in a Java concurrent program, pass CpContext as an AppsContext.

3. Otherwise create AppsContext from the DBC file. If you are running a servlet/JSP in Applications, you can obtain the full path to the DBC file by calling

   ```java
   System.getProperty("JTFDBCFILE") or System.getProperty("BNEDBCFILE")
   ```
Creating Data Source Definition Entries

Add a new data source definition entry to the Template Manager repository as follows:

1. Create an instance of the DataSource class by calling the DataSource.createInstance() method.

2. Set the attributes of the instance.

3. Pass it to the DataSourceHelper.createDataSource() method.

Example

// Create an instance
DataSource d = DataSource.createInstance(ctx, "XDO", "TestDataSource");
// Set properties
d.setDescription("This is the test data source entry.");
d.setStartDate(new java.sql.Date(System.currentTimeMillis()));
d.setName("Test Data Source!");
d.setStatus(TypeDefinitions.DATA_SOURCE_STATUS_ENABLED);
// Call createDataSource() to create an entry into the repository
DataSourceHelper.createDataSource(ctx, d);

Getting and Updating Data Source Definition Entries

Update data source definition entries from the repository by calling the DataSourceHelper.getDataSource() method. It will return an array of DataSource instances. Update these instances by using the data source "set" methods.

Example

// Get data source definition entries
DataSource[] d = DataSourceHelper.getDataSource(ctx, "XDO", "%XDO %");

// Update properties
d.setDescription("New data source entry.");
d.setStartDate(new java.sql.Date(System.currentTimeMillis()));
d.setName("New Data Source name");
d.setStatus(TypeDefinitions.DATA_SOURCE_STATUS_ENABLED);
// Call updateDataSource() to commit the update in the repository
DataSourceHelper.updateDataSource(ctx, d);

Deleting Data Source Definition Entries

Delete data source definition entries by calling the DataSource.deleteDataSource() method. This function does not actually delete the record from the repository, but marks it as "disabled" for future use. You can change the status anytime by calling the DataSource.updateDataSourceStatus() method.

Adding, Updating, and Deleting Schema Files and Sample Files

You can add, update and delete the data source schema definition file and the sample XML file by calling methods defined in the DataSourceHelper class. Please note that unlike the deleteDataSource() method described above, these methods actually delete the schema file and sample records from the repository.
Example
// Add a schema definition file
DataSourceHelper.addSchemaFile(ctx, "XDO", "TestDataSource",
  "schema.xsd", new FileInputStream("/path/to/schema.xsd"));
// Add a sample xml data file
DataSourceHelper.addSampleFile(ctx, "XDO", "TestDataSource",
  "sample.xml", new FileInputStream("/path/to/sample.xml"));

// Update a schema definition file
DataSourceHelper.addSchemaFile(ctx, "XDO", "TestDataSource",
  new FileInputStream("/path/to/new_schema.xsd"));
// Update a sample xml data file
DataSourceHelper.addSampleFile(ctx, "XDO", "TestDataSource",
  new FileInputStream("/path/to/new_sample.xml"));

// Delete a schema definition file
DataSourceHelper.deleteSchemaFile(ctx, "XDO", "TestDataSource");
// Delete a sample xml data file
DataSourceHelper.deleteSampleFile(ctx, "XDO", "TestDataSource");

Getting Schema Files and Sample Files from the Repository
You can download schema files or sample files from the repository by calling the
getSchemaFile() or the getSampleFile() method. These methods return an
InputStream connected to the file contents as a return value.

The sample code is as follows:
Example
// Download the schema definition file from the repository
InputStream schemaFile =
  DataSourceHelper.getSchemaFile(ctx, "XDO", "TestDataSource", );

// Download the XML sample data file from the repository
InputStream sampleFile =
  DataSourceHelper.getSampleFile(ctx, "XDO", "TestDataSource", );

Template APIs
Multiple template objects can be associated with a single data source. The Template class
represents a single template instance. The TemplateHelper class is a utility class used to
create and update template objects in the Template Manager.

The Template Class
The Template class represents a single template object in the template manager. It
is associated with a data source object. The class has several get and set methods to
manipulate the template object.

TemplateHelper Class
The TemplateHelper class is a utility class to manage the template entries in the Template
Manager repository. It consists of a set of static utility methods.
A template entry is a logical unit that represents a single template. Each template entry has a corresponding data source definition entry that defines how the data looks for this template. Each template entry has one physical template file for each language: the locale-specific template files are uploaded separately; and for each translated XLIFF associated with a template, XML Publisher creates and stores a separate XSL file.

Each template entry has one display name and description for each language. These names will be picked up and used when the Template Manager user interface shows the template entry name.

Getting the AppsContext Instance

Some methods require the AppsContext instance to communicate with the Applications database. Get the AppsContext instance in your code using one of the following options:

1. If you are using this class in OA Framework, obtain AppsContext by calling ((OADBTransactionImpl)am.getOADBTransaction()).
   
   where am is your OAApplicationModule.

2. If you are using this class in a Java concurrent program, pass CpContext as an AppsContext.

3. Otherwise create AppsContext from the DBC file. If you are running a servlet/JSP in Applications, you can obtain the full path to the DBC file by calling System.getProperty("JTFDBCFILE") or System.getProperty("BNEDBCFILE")

Getting the OAApplicationModule Instance

Some methods require the OAApplicationModule instance to communicate with the Applications database. Get the OAApplicationModule instance in your code as follows:

1. If you are using the TemplateHelper in OA Framework, you already have an OAApplicationModule instance

2. If you already have AppsContext, you can create the OAApplicationModule instance by using oracle.apps.fnd.framework.server.OAApplicationModuleUtil

It is recommended that you use AppsContext to call APIs because the latest development is based on the APIs that take AppsContext. You can still use APIs that take OAApplicationModule, but they internally call corresponding APIs that take AppsContext.

Creating Template Entries

To add a new template entry to the Template Manager repository:

1. Create an instance of the Template class by calling the Template.createInstance() method

2. Set the attributes of the instance.

3. Pass it to the TemplateHelper.createTemplate() method
Example

// Create an instance
Template t = Template.createInstance(appsContext, "XDO", "TestTemplate",
   TypeDefinitions.TEMPLATE_TYPE_PDF, "XDO", "TestTemplate");

// Set properties
   t.setDescription("This is the test template entry.");
   t.setStartDate(new java.sql.Date(System.currentTimeMillis()));
   t.setName("Test template!");
   t.setStatus(TypeDefinitions.TEMPLATE_STATUS_ENABLED);
// Call createTemplate() to create an entry into the repository
   TemplateHelper.createTemplate(am, t);

Getting and Updating Template Entries

Get template entries from the repository by calling the TemplateHelper.
getTemplate() method or the getTemplates() method. Update the entry information by using use these instances.

Example

// Get active template entries
   Template[] t = TemplateHelper.getTemplates(appsContext, "XDO", "XDO%", true);

// Update properties
   t[0].setDescription("updated template entry.");
   t[0].setStartDate(new java.sql.Date(System.currentTimeMillis()));
   t[0].setName("updated template entry name");
   t[0].setStatus(TypeDefinitions.TEMPLATE_STATUS_ENABLED);
// Call updateTemplate() to commit the update in the repository
   TemplateHelper.updateTemplate(appsContext, t[0]);

Deleting Template Entries

Delete template entries by calling the Template.deleteTemplate() method. The method does not actually delete the record from the repository, but marks it as "disabled" for future use. You can change the status anytime by calling the Template.updateTemplateStatus() method.

Adding, Updating, and Deleting Template Files

You can add, update and delete template files by calling methods defined in the TemplateHelper class. Please note that unlike the template entries, deleting template files actually deletes the record from the repository.

The following code sample demonstrates adding, deleting, and updating a template file:
Example

// Add English template file to the template entry
TemplateHelper.addTemplateFile(
    appsContext, // AppsContext
    "XDO", // Application short name of the template
    "TestTemplate", // Template code of the template
    "en", // ISO language code of the template
    "US", // ISO territory code of the template
    Template.TEMPLATE_TYPE_PDF, // Type of the template file
    "us.pdf", // Filename of the template file
    new FileInputStream("/path/to/us.pdf")); // Template file

// Add Japanese template file to the template entry
TemplateHelper.addTemplateFile(
    appsContext, // AppsContext
    "XDO", // Application short name of the template
    "TestTemplate", // Template code of the template
    "ja", // ISO language code of the template
    "JP", // ISO territory code of the template
    Template.TEMPLATE_TYPE_PDF, // Type of the template file
    "ja.pdf", // Filename of the template file
    new FileInputStream("/path/to/ja.pdf")); // Template file

// Update English template file to the template entry
TemplateHelper.updateTemplateFile(
    appsContext, // AppsContext
    "XDO", // Application short name of the template
    "TestTemplate", // Template code of the template
    "en", // ISO language code of the template
    "US", // ISO territory code of the template
    Template.TEMPLATE_TYPE_PDF, // Type of the template file
    "us.pdf", // Filename of the template file
    new FileInputStream("/path/to/new/us.pdf")); // Template file

// Delete Japanese template file to the template entry
TemplateHelper.deleteTemplateFile(
    appsContext, // AppsContext
    "XDO", // Application short name of the template
    "TestTemplate", // Template code of the template
    "ja", // ISO language code of the template
    "JP")); // ISO territory code of the template

Getting Template Files

Download template file contents from the repository by calling the getTemplateFile() methods. These methods return an InputStream connected to the template file as a return value.

Example

// Download the English template file from the repository
InputStream in = TemplateHelper.getTemplateFile(
    appsContext, // AppsContext
    "XDO", // Application short name of the template
    "TestTemplate", // Template code of the template
    "en", // ISO language code of the template
    "US")); // ISO territory code of the template
Processing Templates

You can apply a template, stored in the Template Manager, to an XML data source by calling one of the processTemplate() methods. You need to pass the OutputStream object for the destination of the processed document.

Example

```java
// Process template
TemplateHelper.processTemplateFile(
    appsContext,  // AppsContext
    "XDO",       // Application short name of the template
    "TestTemplate", // Template code of the template
    "en",        // ISO language code of the template
    "US",        // ISO territory code of the template
dataInputStream,  // XML data for the template
TemplateHelper.OUT_TYPE_PDF,  // Output type of the processed document
properties,        // Properties for the template processing
docOutputStream)  // OutputStream where the processed document goes.
```

Pass the properties for template processing by passing a Properties object. You can pass null if you have no properties to pass to the XML Publisher processors.

Passing XSL Parameters to RTF/FO Templates:

1. Set the parameter names and values in a Properties object.
   
   All property names for RTF/XSL templates must start with "xslt."

   The parameter value must be in single quotes.

   Example

   ```java
   String <parameter_name> = "name";
   String <parameter_value> = "value";
   Properties props = new Properties();
   ...
   props.put("xslt.<parameter_name>", "'<parameter_value>'");
   ...
   String <parameter_name> = "name";
   ```

2. Set this properties object when you call a TemplateHelper.processTemplate() method.

   Example

   ```java
   TemplateHelper.processTemplate(appsContext, "OKC", "OKCTERMS", "en", "US", pData, props, pOutput);
   ```

3. Define XSL parameters in an RTF/FO template using the following syntax:

   ```xml
   <xsl:param name="parameter_name" select "default_value" xdofo:ctx ="begin"/>
   ```

In addition to passing the properties that you set, the TemplateHelper class also looks up the following locations to get system level properties if available:

1. Java system properties for OA specific properties, such as the OA_MEDIA location.
2. System configuration file located at {java.home}/lib/xdo.cfg
If there are conflicts between system level properties and user level properties that you pass, user level properties will take precedence.

Creating and Processing EFT/EDI Templates

The TemplateHelper class supports EFT/EDI templates. You can create EFT/EDI template entries with Template.TEMPLATE_TYPE_ETEXT template type. You can also process the EFT/EDI templates by using the processTemplate() method in the TemplateHelper. You can assign OUTPUT_TYPE_ETEXT output type when you process EFT/EDI templates. If you need to supply parameters to the EFT/EDI processing engine, you can pass those parameters as a Properties object when you call the processTemplate() method.

Example

```java
// Process EFT/EDI template
TemplateHelper.processTemplateFile(
    appsContext,  // AppsContext
    "XDO",  // Application short name of the template
    "TestTemplate",  // Template code of the template
    "en",  // ISO language code of the template
    "US",  // ISO territory code of the template
    dataInputStream,  // XML data for the template
    TemplateHelper.OUTPUT_TYPE_ETEXT,  // Output type of the processed document properties,  // Properties for the template processing.
    docOutputStream)  // OutputStream where the processed document goes.
```

If you need more control for EFT/EDI template processing (such as for getting/setting context parameters for the EFT/EDI processing engine), you can call EFTGenerator to process templates.

Example

```java
import oracle.apps.xdo.template.eft.EFTGenerator;

...

// Process EFT/EDI template with EFTGenerator class
EFTGenerator generator = new EFTGenerator();
// Get the template file from template manager repository
// and set it.
generator.loadXSL(
    TemplateHelper.getTemplateFile(ctx, "XDO", "TestTemplate", "en",
    "US");
// Set the data XML
generator.loadXML(dataInputStream);
// Set context param
generator.setContextParam(PARAM1, PARAM1_VALUE);
// Process the template
generator.process(resultOutputStream);
// Get context param
String param2 = generator.getContextParam(PARAM2);
```
Language Fallback Mechanism

Both the `getTemplateFile()` and the `processTemplate()` methods support the language fallback mechanism. This mechanism provides the most appropriate InputStream even if there is no template file to match the language criteria. The priority of the language fallback is as follows:

1. Returns the template file that matches the given language and territory.
2. Returns the template file that matches the given language and is territory independent (the territory value is "00").
3. Returns the default template. See The Default Template, page 5-5 for more information on assigning a default template file.

For example, the following table shows a sample of templates in the Template Manager repository:

<table>
<thead>
<tr>
<th>Template File</th>
<th>ISO Language Code</th>
<th>ISO Territory Code</th>
<th>Default?</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>en</td>
<td>US</td>
<td>no</td>
</tr>
<tr>
<td>B</td>
<td>en</td>
<td>00</td>
<td>no</td>
</tr>
<tr>
<td>C</td>
<td>fr</td>
<td>FR</td>
<td>yes</td>
</tr>
<tr>
<td>D</td>
<td>ja</td>
<td>JP</td>
<td>no</td>
</tr>
</tbody>
</table>

The following table shows the template that will be returned if you pass the given ISO language/territory code combinations:

<table>
<thead>
<tr>
<th>ISO Language Code</th>
<th>ISO Territory Code</th>
<th>Template Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>en</td>
<td>US</td>
<td>A</td>
</tr>
<tr>
<td>en</td>
<td>GB</td>
<td>B</td>
</tr>
<tr>
<td>en</td>
<td>null</td>
<td>B</td>
</tr>
<tr>
<td>fr</td>
<td>FR</td>
<td>C</td>
</tr>
<tr>
<td>ja</td>
<td>JP</td>
<td>D</td>
</tr>
<tr>
<td>de</td>
<td>DE</td>
<td>C</td>
</tr>
</tbody>
</table>

It is recommended that you pass both the ISO language code and territory code explicitly to best obtain the target template file.

Template Validation

By default, when you call `getTemplateFile()` or `processTemplate()`, XML Publisher runs validation logic against START_DATE, END_DATE, and TEMPLATE_STATUS set in the template entry. If an invalid entry is found, the following exceptions are thrown accordingly: TemplateExpiredException, TemplateNotYetValidException, StatusDisabledException. These exceptions are subclasses of the oracle.apps.xdo.XDOException so you can catch XDOException if you want to catch all these exceptions at one time. To turn off this validation mode, set the java system property `xdo.TemplateValidation=false`. The default mode is set to true.
Translatable Templates

You can define a translatable template for each template code. The text in the template file can be retrieved in XLIFF format to be translated and merged back into the file, using the `getXLIFF()` and `uploadXLIFF()` methods.

- `getXLIFF()` - Downloads the translatable boilerplate text for a template in xlfiff format. A translatable template file must exist for this template, else the return value will be null. The specified locale will be added to the target-language attribute of the resulting document. If translations are not available for this locale, the resulting xlfiff document will have empty elements.

- `uploadXLIFF()` - Uploads the translations for a template in xlfiff format. The xlfiff file must contain a valid target-language attribute.
Introduction

The Delivery Manager is a set of Java APIs that allow you to control the delivery of your XML Publisher documents. Use the Delivery Manager to:

- Deliver documents through established delivery channels (e-mail, fax, printer, WebDAV, FTP, Secure FTP, AS2, or HTTP) or custom delivery channels
- Track the status of each delivery
- Redeliver documents

Using the Delivery Manager

To use the Delivery Manager follow these steps:

1. Create a DeliveryManager instance
2. Create a DeliveryRequest instance using the createRequest() method
3. Add the request properties (such as DeliveryRequest destination). Most properties require a String value. See the supported properties for each delivery channel for more information.
4. Set your document to the DeliveryRequest.
5. Call submit() to submit the delivery request.

One delivery request can handle one document and one destination. This facilitates monitoring and resubmission, if necessary.

DeliveryRequest allows you to set the documents in three ways as follows:

- Get OutputStream from the DeliveryRequest and write the document to the OutputStream. You do not need to close the OutputStream to call the submit() method immediately after you finish writing the document to the OutputStream.
- Set InputStream of the document to DeliveryRequest. The DeliveryRequest will read the InputStream when you call submit() for the first time. The DeliveryRequest does not close the InputStream so you must ensure to close it.
- Set the file name of the document to DeliveryRequest.

The Delivery Manager supports streamlined delivery when you set the direct mode. See Direct and Buffering Modes, page 11-24.

The follow delivery channels are described in this document:
Delivering Documents via e-Mail

The following sample demonstrates delivery via E-mail:

**Example**

```java
// create delivery manager instance
DeliveryManager dm = new DeliveryManager();
// create a delivery request
DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_SMTP_EMAIL);

// set email subject
req.addProperty(DeliveryPropertyDefinitions.SMTP_SUBJECT, "test mail");
// set SMTP server host
req.addProperty(DeliveryPropertyDefinitions.SMTP_HOST, "mysmtphost");
// set the sender email address
req.addProperty(DeliveryPropertyDefinitions.SMTP_FROM, "mynam

e@mydomain.com");
// set the destination email address
req.addProperty(DeliveryPropertyDefinitions.SMTP_TO_RECIPIENTS, "user1@mydo
main.com, user2@mydomain.com" );
// set the content type of the email body
req.addProperty(DeliveryPropertyDefinitions.SMTP_CONTENT_TYPE , "application/pdf");
// set the document file name appeared in the email
req.addProperty(DeliveryPropertyDefinitions.SMTP_CONTENT_FILE
NAME, "test.pdf");
// set the document to deliver
req.setDocument("/document/test.pdf");

// submit the request
req.submit();
// close the request
req.close();
```

The following table lists the supported properties:
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
</table>
| SMTP_TO_RECIPIENTS       | Required
Enter multiple recipients separated by a comma (example: "user1@mydomain.com, user2@mydomain.com") |
| SMTP_CC_RECIPIENTS       | Optional
Enter multiple recipients separated by a comma.                                                                                         |
| SMTP_BCC_RECIPIENTS      | Optional
Enter multiple recipients separated by a comma.                                                                                         |
| SMTP_FROM                | Required
Enter the e-mail address of the sending party.                                                                                             |
| SMTP_REPLY_TO            | Optional
Enter the reply-to e-mail address.                                                                                                         |
| SMTP_SUBJECT             | Required
Enter the subject of the e-mail.                                                                                                          |
| SMTP_CHARACTER_ENCODING  | Optional
Default is "UTF-8".                                                                                                                          |
| SMTP_ATTACHMENT          | Optional
If you are including an attachment, enter the attachment object name.                                                                        |
| SMTP_CONTENT_FILENAME    | Required
Enter the file name of the document (example: invoice.pdf)                                                                                   |
| SMTP_CONTENT_TYPE        | Required
Enter the MIME type.                                                                                                                         |
| SMTP_SMTP_HOST           | Required
Enter the SMTP host name.                                                                                                                   |
| SMTP_SMTP_PORT           | Optional
Enter the SMTP port. Default is 25.                                                                                                          |
| SMTP_SMTP_USERNAME       | Optional
If the SMTP server requires authentication, enter your username for the server.                                                             |
| SMTP_SMTP_PASSWORD       | Optional
If the SMTP server requires authentication, enter the password for the username you entered.                                               |
| SMTP_ATTACHMENT_FIRST    | Optional
If your e-mail contains an attachment and you want the attachment to appear first, enter "true". If you do not want the attachment to appear first, enter "false". |
Defining Multiple Recipients

The e-mail delivery server channel supports multiple documents and multiple destinations per request. The following example demonstrates multiple TO and CC addresses:

Example

// set the TO email addresses
req.addProperty(
    DeliveryPropertyDefinitions.SMTP_TO_RECIPIENTS,
    "user1@mydomain.com", user2@mydomain.com, user3@mydomain.com"};

// set the CC email addresses
req.addProperty(
    DeliveryPropertyDefinitions.SMTP_CC_RECIPIENTS,
    "user4@mydomain.com, user5@mydomain.com, user6@mydomain.com"};

Attaching Multiple Documents into One Request

Use the Attachment utility class to attach multiple documents into one request. Sample usage is as follows:
Example

// Properties for Attachment
Hashtable props = new Hashtable();
// Set encoding property for the non-ASCII file names.
// It’s optional. Default value is "UTF-8"
props.put(DeliveryPropertyDefinitions.SMTP_CHARACTER_ENCODING, "UTF-8");

(You can append other properties also)

// create Attachment instance
Attachment m = new Attachment(props);

// add attachment files
m.addAttachment("/pdf_sample/pdfTest5.pdf", "a1.pdf", "application/pdf");
m.addAttachment("/rtf_sample/rtfsample_en00.rtf", "a2.rtf", "application/rtf");
m.addAttachment("/xml_sample/pdfTest5.xml", "a3.xml", "text/xml");

req.addProperty(DeliveryPropertyDefinitions.SMTP_ATTACHMENT, m);

---

Attaching HTML Documents

You can attach HTML documents into one request. If you have references to image files located in the local file system in your HTML document, the Attachment utility automatically attaches those image files also. The sample usage is as follows:

Example

Attachment m = new Attachment();
m.addHtmlAttachment("/path/to/my.html");

req.addProperty(DeliveryPropertyDefinitions.SMTP_ATTACHMENT, m);
Displaying the Attachment at the top of the e-mail

If you want to show your attachment at the top of the e-mail, set the property SMTP_ATTACHMENT_FIRST to "true". Sample usage is as follows.

**Example**

```java
Attachment m = new Attachment();
m.addHtmlAttachment("/path/to/my.html");

req.addProperty(DeliveryPropertyDefinitions.SMTP_ATTACHMENT_FIRST, "true");
```

Providing Username and Password for Authentication

If the SMTP server requires authentication, you can specify the username and password to the delivery request.

**Example**

```java

req.addProperty(DeliveryPropertyDefinitions.SMTP_USERNAME, "scott");
req.addProperty(DeliveryPropertyDefinitions.SMTP_PASSWORD, "tig er");
```

Delivering Your Document to a Printer

The Delivery Server supports Internet Printing Protocol (IPP) as defined in RFC 2910 and 2911 for the delivery of documents to IPP-supported printers or servers, such as CUPS.

Common Unix Printing System (CUPS) is a free, server-style, IPP-based software that can accept IPP requests and dispatch those requests to both IPP and non-IPP based devices, such as printers and fax machines. See [http://www.cups.org/](http://www.cups.org/) for more information about CUPS. See Setting Up Cups, page 11-36 for additional information about setting up CUPS in your system.

Following is a code sample for delivery to a printer:
**Example**

```java
// create delivery manager instance
DeliveryManager dm = new DeliveryManager();
// create a delivery request
DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_IPP_PRINTER);

// set IPP printer host
req.addProperty(DeliveryPropertyDefinitions.IPP_HOST, "myhost");

// set IPP printer port
req.addProperty(DeliveryPropertyDefinitions.IPP_PORT, "631");
// set IPP printer name
req.addProperty(DeliveryPropertyDefinitions.IPP_PRINTER_NAME, "/printers/myprinter");
// set the document format
req.addProperty(DeliveryPropertyDefinitions.IPP_DOCUMENT_FORMAT, DeliveryPropertyDefinitions.IPP_DOCUMENT_FORMAT_POSTSCRIPT);

// set the document
req.setDocument("/document/invoice.ps");

// submit the request
req.submit();
// close the request
req.close();
```

The following properties are supported. A string value is required for each property, unless otherwise noted. Note that printer-specific properties such as IPP_SIDES, IPP_COPIES and IPP_ORIENTATION depend on the printer capabilities. For example, if the target printer does not support duplex printing, the IPP_SIDES setting will have no effect.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPP_HOST</td>
<td>Required&lt;br&gt;Enter the host name.</td>
</tr>
<tr>
<td>IPP_PORT</td>
<td>Optional&lt;br&gt;Default is 631.</td>
</tr>
<tr>
<td>IPP_PRINTER_NAME</td>
<td>Required&lt;br&gt;Enter the name of the printer that is to receive the output (example: /printers/myPrinter).</td>
</tr>
<tr>
<td>IPP_AUTHTYPE</td>
<td>Optional&lt;br&gt;Valid values for authentication type are:&lt;br&gt;IPP_AUTHTYPE_NONE - no authentication (default)&lt;br&gt;IPP_AUTHTYPE_BASIC - use HTTP basic authentication&lt;br&gt;IPP_AUTHTYPE_DIGEST - use HTTP digest authentication</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IPP_USERNAME</td>
<td>Optional Enter the username for HTTP authentication.</td>
</tr>
<tr>
<td>IPP_PASSWORD</td>
<td>Optional Enter the password for HTTP authentication.</td>
</tr>
<tr>
<td>IPP_ENCTYPE</td>
<td>Optional The encryption type can be set to either of the following:IPP_ENCTYPE_NONE - no encryption (default)IPP_ENCTYPE_SSL - use Secure Socket Layer</td>
</tr>
<tr>
<td>IPP_USE_FULL_URL</td>
<td>Optional Set to &quot;true&quot; to send the full URL for the HTTP request header. Valid values are &quot;true&quot; or &quot;false&quot; (default).</td>
</tr>
<tr>
<td>IPP_USE_CHUNKED_BODY</td>
<td>Optional Valid values are &quot;true&quot; (default) to use HTTP chunked transfer coding for the message body, or &quot;false&quot;.</td>
</tr>
<tr>
<td>IPP_ATTRIBUTE_CHARSET</td>
<td>Optional Attribute character set of the IPP request. Default is &quot;UTF-8&quot;.</td>
</tr>
<tr>
<td>IPP_NATURAL_LANGUAGE</td>
<td>Optional The natural language of the IPP request. Default is &quot;en&quot;.</td>
</tr>
<tr>
<td>IPP_JOB_NAME</td>
<td>Optional Job name of the IPP request.</td>
</tr>
<tr>
<td>IPP_COPIES</td>
<td>Optional Define the number of copies to print (example: &quot;1&quot;, &quot;5&quot;, &quot;10&quot;). Default is 1.</td>
</tr>
</tbody>
</table>
| IPP_SIDES                    | Optional Enable two-sided printing. This setting will be ignored if the target printer does not support two-sided printing. Valid values are:IPP_SIDES_ONE_SIDED - defaultIPP_SIDES_TWO_SIDED_LONG_EDGE - prints both sides of paper for binding long edge.IPP_SIDES_TWO_SIDED_SHORT_EDGE - prints both sides of paper for binding short edge.
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPP_ORIENTATIONS</td>
<td>Optional Sets the paper orientation. This setting will be ignored if the target printer does not support orientation settings. Valid values are:</td>
</tr>
<tr>
<td></td>
<td>IPP_ORIENTATIONS_PORTRAIT (default)</td>
</tr>
<tr>
<td></td>
<td>IPP_ORIENTATIONS_LANDSCAPE</td>
</tr>
<tr>
<td>IPP_DOCUMENT_FORMAT</td>
<td>Optional The target printer must support the specified format. Valid values are:</td>
</tr>
<tr>
<td></td>
<td>IPP_DOCUMENT_FORMAT_POSTSCRIPT</td>
</tr>
<tr>
<td></td>
<td>IPP_DOCUMENT_FORMAT_PLAINTEXT</td>
</tr>
<tr>
<td></td>
<td>IPP_DOCUMENT_FORMAT_PDF</td>
</tr>
<tr>
<td></td>
<td>IPP_DOCUMENT_FORMAT_OCTETSTREAM (default)</td>
</tr>
</tbody>
</table>

**Printing over an HTTP Proxy Server**

To deliver documents to IPP printers or fax machines over an HTTP proxy server, you may encounter delivery problems due to differences in the HTTP implementations between CUPS and the proxy servers. Setting the following two properties can resolve most of these problems:

- DeliveryPropertyDefinitions.IPP_USE_FULL_URL - set to "true"
- DeliveryPropertyDefinitions.IPP_USE_CHUNKED_BODY - set to "false"

**Delivering Your Documents via Fax**

The delivery system supports the delivery of documents to fax modems configured on CUPS. You can configure fax modems on CUPS with efax (http://www.cce.com/efax/) and FAX4CUPS (http://www.gnu.orgirectory/productivity/special/fax4CUPS.html).

Sample code for fax delivery is as follows:
**Example**

// create delivery manager instance
DeliveryManager dm = new DeliveryManager();
// create a delivery request
DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_IPP_FAX);

// set IPP fax host
req.addProperty(DeliveryPropertyDefinitions.IPP_HOST, "myhost");
// set IPP fax port
req.addProperty(DeliveryPropertyDefinitions.IPP_PORT, "631");
// set IPP fax name
req.addProperty(DeliveryPropertyDefinitions.IPP_PRINTER_NAME, "/printers/myfax");
// set the document format
req.addProperty(DeliveryPropertyDefinitions.IPP_DOCUMENT_FORMAT, "application/postscript");
// set the phone number to send
req.addProperty(DeliveryPropertyDefinitions.IPP_PHONE_NUMBER, "9999999");
// set the document
req.setDocument("/document/invoice.pdf");

// submit the request
req.submit();
// close the request
req.close();

The supported properties are the same as those supported for printer documents, plus the following:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPP_PHONE_NUMBER</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td>Enter the fax number.</td>
</tr>
</tbody>
</table>

**Delivering Your Documents to WebDAV Servers**

The following is sample code for delivery to a WebDAV server:
Example
// create delivery manager instance
    DeliveryManager dm = new DeliveryManager();
// create a delivery request
    DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_WEBDAV);

    // set document content type
    req.addProperty(DeliveryPropertyDefinitions.WEBDAV_CONTENT_TYPE, "application/pdf");
// set the WebDAV server hostname
    req.addProperty(DeliveryPropertyDefinitions.WEBDAV_HOST, "mywebdavhost");
// set the WebDAV server port number
    req.addProperty(DeliveryPropertyDefinitions.WEBDAV_PORT, "80");
// set the target remote directory
    req.addProperty(DeliveryPropertyDefinitions.WEBDAV_REMOTE_DIRECTORY, "/content/");
// set the remote filename
    req.addProperty(DeliveryPropertyDefinitions.WEBDAV_REMOTE_FILENAME, "xdotest.pdf");

    // set username and password to access WebDAV server
    req.addProperty(DeliveryPropertyDefinitions.WEBDAV_USERNAME, "xdo");
    req.addProperty(DeliveryPropertyDefinitions.WEBDAV_PASSWORD, "xdo");
// set the document
    req.setDocument("/document/test.pdf");

    // submit the request
    req.submit();
// close the request
    req.close();

The following properties are supported. A String value is required for each, unless otherwise noted.
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
</table>
| WEBDAV_CONTENT_TYPE      | Required  
  Enter the document content type (example: "application/pdf").                                                                         |
| WEBDAV_HOST              | Required  
  Enter the server host name.                                                                                                              |
| WEBDAV_PORT              | Optional  
  Enter the server port number.  
  Default is 80.                                                                                                                                  |
| WEBDAV_REMOTE_DIRECTORY  | Required.  
  Enter the remote directory name (example: "/myreports/").                                                                                 |
| WEBDAV_REMOTE_FILENAME   | Required.  
  Enter the remote file name.                                                                                                               |
| WEBDAV_AUTHTYPE          | Optional  
  Valid values for authentication type are:                                                                          
  WEBDAV_AUTHTYPE_NONE - no authentication (default)                                                                                           
  WEBDAV_AUTHTYPE_BASIC - use HTTP basic authentication                                                                                       
  WEBDAV_AUTHTYPE_DIGEST - use HTTP digest authentication                                                                                     |
| WEBDAV_USERNAME          | Optional  
  Enter the username for HTTP authentication.                                                                                               |
| WEBDAV_PASSWORD          | Optional  
  Enter the password for HTTP authentication.                                                                                              |
| WEBDAV_ENCTYPE           | Optional  
  Valid values for encryption type are:                                                                                                  
  WEBDAV_ENCTYPE_NONE - no encryption (default)                                                                                               
  WEBDAV_ENCTYPE_SSL - use Secure Socket Layer                                                                                                 |
| WEBDAV_USE_FULL_URL      | Optional  
  Set to "true" to send the full URL for the HTTP request header. Valid values are "true" or "false" (default).                             |
| WEBDAV_USE_CHUNKED_BODY  | Optional  
  Valid values are "true" (default) to use HTTP chunked transfer coding for the message body, or "false".                                 |
Deliver Your Documents Using FTP

The following is sample code for delivery to a FTP server:

**Example**

```java
// create delivery manager instance
DeliveryManager dm = new DeliveryManager();
// create a delivery request
DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_FTP);

// set hostname of the FTP server
req.addProperty(DeliveryPropertyDefinitions.FTP_HOST, "myftphost");
// set port# of the FTP server
req.addProperty(DeliveryPropertyDefinitions.FTP_PORT, "21");
// set username and password to access WebDAV server
req.addProperty(DeliveryPropertyDefinitions.FTP_USERNAME, "xdoo");
req.addProperty(DeliveryPropertyDefinitions.FTP_PASSWORD, "xdoo");
// set the remote directory that you want to send your document to
req.addProperty(DeliveryPropertyDefinitions.FTP_REMOTE_DIRECTORY, "pub");
// set the remote file name
req.addProperty(DeliveryPropertyDefinitions.FTP_REMOTE_FILENAME, "test.pdf");
// set the document
req.setDocument("/document/test.pdf");

// submit the request
req.submit();
// close the request
req.close();
```

The following properties are supported. A String value is required unless otherwise noted.
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTP_HOST</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td>Enter the server host name.</td>
</tr>
<tr>
<td>FTP_PORT</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Enter the server port number. Default is 21.</td>
</tr>
<tr>
<td>FTP_USERNAME</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td>Enter the login user name to the FTP server.</td>
</tr>
<tr>
<td>FTP_PASSWORD</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td>Enter the login password to the FTP server.</td>
</tr>
<tr>
<td>FTP_REMOTE_DIRECTORY</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td>Enter the directory to which to deliver the</td>
</tr>
<tr>
<td></td>
<td>document (example: /pub/)</td>
</tr>
<tr>
<td>FTP_REMOTE_FILENAME</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td>Enter the document file name for the remote</td>
</tr>
<tr>
<td></td>
<td>server.</td>
</tr>
<tr>
<td>FTP_BINARY_MODE</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values are &quot;true&quot; (default) or &quot;false&quot;.</td>
</tr>
</tbody>
</table>

**Delivering Documents over Secure FTP**

Secure FTP is the protocol based on the Secure Shell technology (ssh) and it is widely used to transfer files in a secure manner. Both Secure Shell and Secure FTP are defined by the Internet Engineering Task Force (IETF) and the specifications are available on their Web site: http://www.ietf.org. The delivery system supports the delivery of documents to secure FTP servers.

The following tables lists the supported properties. A string value is required for each property unless otherwise noted.
Example
// create delivery manager instance
DeliveryManager dm = new DeliveryManager();
// create a delivery request
DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_SFTP);
// set hostname of the SFTP server
req.addProperty(DeliveryPropertyDefinitions.SFTP_HOST, "mysftphost");
// set username and password to access server
req.addProperty(DeliveryPropertyDefinitions.SFTP_USERNAME, "myname");
req.addProperty(DeliveryPropertyDefinitions.SFTP_PASSWORD, "mypassword");
// set the remote directory that you want to send your document to
req.addProperty(DeliveryPropertyDefinitions.SFTP_REMOTE_DIRECTORY, "pub");
// set the remote file name
req.addProperty(DeliveryPropertyDefinitions.SFTP_REMOTE_FILENAME, "test.pdf");
// set the document
req.setDocument("/document/test.pdf");

// submit the request
req.submit();
// close the request
req.close();
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFTP_HOST</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td>Enter the target server host name.</td>
</tr>
<tr>
<td>SFTP_PORT</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Enter the target server SSH port number. Default is 22.</td>
</tr>
<tr>
<td>SFTP_USERNAME</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td>Enter the login user name.</td>
</tr>
<tr>
<td>SFTP_PASSWORD</td>
<td>Required if you choose the SFTP_AUTH_TYPE_PASSWORD authentication type.</td>
</tr>
<tr>
<td></td>
<td>Enter the login password.</td>
</tr>
<tr>
<td>SFTP_REMOTE_DIRECTORY</td>
<td>Enter the directory to which to deliver the document (example: /pub). If no value is entered, the document will be delivered to the login directory.</td>
</tr>
<tr>
<td>SFTP_REMOTE_FILENAME</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td>Enter the document file name on the remote server.</td>
</tr>
<tr>
<td>SFTP_AUTH_TYPE</td>
<td>Set either of the following:</td>
</tr>
<tr>
<td></td>
<td>SFTP_AUTH_TYPE_PASSWORD (Default) Requires providing password at login.</td>
</tr>
<tr>
<td></td>
<td>SFTP_AUTH_TYPE_PUBLIC_KEY - public key authorization type.</td>
</tr>
<tr>
<td>SFTP_PRIVATE_KEY_FILE</td>
<td>Enter the client private key file. Required if you choose SFTP_AUTH_TYPE_PUBLIC_KEY.</td>
</tr>
<tr>
<td>SFTP_PRIVATE_KEY_PASSWORD</td>
<td>Enter the client private key password. Required if you choose SFTP_AUTH_TYPE_PUBLIC_KEY.</td>
</tr>
<tr>
<td>SFTP_FILE_PERMISSION</td>
<td>Enter the permissions to set for the file being created. Default is 0755.</td>
</tr>
</tbody>
</table>

**Authentication Modes**

The secure FTP delivery supports two authentication modes: password authentication and public key authentication. Set the property SFTP_AUTH_TYPE to choose the mode. The default mode is password authentication.
The password authentication mode requires the username and password to log in to the secure FTP server. The following example shows sample code:

Example

```
// set password auth type
req.addProperty(DeliveryPropertyDefinitions.SFTP_AUTH_TYPE, DeliveryPropertyDefinitions.SFTP_AUTH_TYPE_PASSWORD);
// set username and password to access server
req.addProperty(DeliveryPropertyDefinitions.SFTP_USERNAME, "mynname");
req.addProperty(DeliveryPropertyDefinitions.SFTP_PASSWORD, "mypassword");
```

The public key authorization mode requires the username, your private key and password for the private key. This is a more secure method than the password authentication. Note that in order to use the public key authentication mode, you must set up the public key in the ssh/secure FTP server in advance. The following example shows sample code:
Delivering Documents over HTTP

The Delivery Manager supports delivery of documents to HTTP servers. The following sample sends a document through the HTTP POST method. Note that the receiving HTTP server must be able to accept your custom HTTP request in advance (for example via a custom servlet or CGI program).
Example
// create delivery manager instance
DeliveryManager dm = new DeliveryManager();
// create a delivery request
DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_HTTP);

   // set request method
   req.addProperty(DeliveryPropertyDefinitions.HTTP_METHOD, DeliveryPropertyDefinitions.HTTP_METHOD_POST);
   // set document content type
   req.addProperty(DeliveryPropertyDefinitions.HTTP_CONTENT_TYPE, "application/pdf");
   // set the HTTP server hostname
   req.addProperty(DeliveryPropertyDefinitions.HTTP_HOST, "myhost");
   // set the HTTP server port number
   req.addProperty(DeliveryPropertyDefinitions.HTTP_PORT, "80");
   // set the target remote directory
   req.addProperty(DeliveryPropertyDefinitions.HTTP_REMOTE_DIRECTORY, "/servlet/");
   // set the remote filename (servlet class)
   req.addProperty(DeliveryPropertyDefinitions.HTTP_REMOTE_FILE_NAME, "uploadDocument");

   // set the document
   req.setDocument("/document/test.pdf");

   // submit the request
   req.submit();
   // close the request
   req.close();

The following table lists the properties that are supported. A String value is required for each property unless otherwise noted.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_METHOD</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Sets the HTTP request method. Valid values are: HTTP_METHOD_POST (Default) HTTP_METHOD_PUT</td>
</tr>
<tr>
<td>HTTP_CONTENT_TYPE</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>The document content type (example: &quot;application/pdf&quot;).</td>
</tr>
<tr>
<td>HTTP_HOST</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td>Enter the server host name.</td>
</tr>
<tr>
<td>HTTP_PORT</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Enter the server port number. The default is 80.</td>
</tr>
<tr>
<td>HTTP_REMOTE_DIRECTORY</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td>Enter the remote directory name (example: &quot;/home/&quot;).</td>
</tr>
</tbody>
</table>
### Property Description

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
</table>
| HTTP_REMOTE_FILENAME           | Required  
  Enter the file name to save the document as in the remote directory. |
| HTTP_AUTHTYPE                  | Optional  
  Valid values for authentication type are:  
  HTTP_AUTHTYPE_NONE - no authentication (default)  
  HTTP_AUTHTYPE_BASIC - use basic HTTP authentication  
  HTTP_AUTHTYPE_DIGEST - use digest HTTP authentication |
| HTTP_USERNAME                  | Optional  
  If the server requires authentication, enter the username. |
| HTTP_PASSWORD                  | Optional  
  If the server requires authentication, enter the password for the username. |
| HTTP_ENCTYPE                   | Optional  
  Enter the encryption type:  
  HTTP_ENCTYPE_NONE - no encryption (default)  
  HTTP_ENCTYPE_SSL - use Secure Socket Layer |
| HTTP_USE_FULL_URL              | Optional  
  Set to "true" to send the full URL for the HTTP request header. Valid values are "true" or "false" (default). |
| HTTP_USE_CHUNKED_BODY          | Optional  
  Valid values are "true" (default) to use HTTP chunked transfer coding for the message body, or "false". |
| HTTP_TIMEOUT                   | Optional  
  Enter a length of time in milliseconds after which to terminate the request if a connection is not made to the HTTP server. The default is 60000 (1 minute). |

---

**Delivering Documents via AS2**

AS2 is one of the standard protocols defined in the Electronic Data Interchange-Internet Integration (EDI-INT). AS2 is based on HTTP and other internet standard technologies and is designed to exchange data over the internet in a secure manner. The AS2 specification is defined in RFC4130 (available at http://www.ietf.org/). The delivery system supports the delivery of documents to AS2 servers. Sample code is as follows:
Example
// create delivery manager instance
DeliveryManager dm = new DeliveryManager();
// create a delivery request
DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_AS2);

// set AS2 message properties
req.addProperty(DeliveryPropertyDefinitions.AS2_FROM, "Me");
req.addProperty(DeliveryPropertyDefinitions.AS2_TO, "You");
req.addProperty(DeliveryPropertyDefinitions.AS2_SUBJECT, "My EDI Message");
req.addProperty(DeliveryPropertyDefinitions.AS2_CONTENT_TYPE, "applications/EDIFACT");

// set HTTP properties
req.addProperty(DeliveryPropertyDefinitions.AS2_HTTP_HOST, "as2host");
req.addProperty(DeliveryPropertyDefinitions.AS2_HTTP_REMOTE_DIRECTORY, "/");
req.addProperty(DeliveryPropertyDefinitions.AS2_HTTP_REMOTE_FILENAME, "as2");

// set the document
req.setDocument("/document/myEDIdoc");
// submit the request
DeliveryResponse res = req.submit();
// close the request
req.close();

The following table lists the supported properties. A string value is required for each property unless otherwise noted.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS2_FROM</td>
<td>Required. Enter the AS2 message sender.</td>
</tr>
<tr>
<td>AS2_TO</td>
<td>Required. Enter the AS2 message recipient.</td>
</tr>
<tr>
<td>AS2_SUBJECT</td>
<td>Required. Enter the message subject.</td>
</tr>
<tr>
<td>AS2_MESSAGE_COMPRESSION</td>
<td>Default value is False. Enter True to compress the message.</td>
</tr>
<tr>
<td>AS2_MESSAGE_SIGNATURE</td>
<td>Default value is False. Enter True to sign the message.</td>
</tr>
<tr>
<td>AS2_MESSAGE_ENCRYPTION</td>
<td>Default value is False. Enter True to encrypt the message.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| AS2_CONTENT_TYPE               | Required. Enter the content type of the document. Valid values are:  
  • application/EDIFACT  
  • application/xml |
| AS2_ENC_ALGO                   | The AS2 encryption algorithm. Set one of the following:  
  • AS2_ENC_ALGO_RC2_40  
  • AS2_ENC_ALGO_RC2_64  
  • AS2_ENC_ALGO_RC2_128  
  • AS2_ENC_ALGO_DES  
  • AS2_ENC_ALGO_DES_EDE3 (Default)  
  • AS2_ENC_ALGO_AES_128  
  • AS2_ENC_ALGO_AES_192  
  • AS2_ENC_ALGO_AES_256 |
| AS2_DIGEST_ALGO                | Enter the AS2 digest algorithm for signing the messages. Set either of the following:  
  • AS2_DIGEST_ALGO_MD5 (Default)  
  • AS2_DIGEST_ALGO_SHA1 |
| AS2_ASYNC_ADDRESS              | Enter the asynchronous address to which MDN notifications should be set. |
| AS2_ASYNC_EMAIL_SERVER_HOST    | Enter the email server host for asynchronous email MDN. |
| AS2_ASYNC_EMAIL_SERVER_PORT    | Enter the email server port for asynchronous email MDN. |
| AS2_ASYNC_EMAIL_SERVER_USERNAME| Enter the email server USERNAME for asynchronous email MDN. |
| AS2_ASYNC_EMAIL_SERVER_PASSWORD| Enter the email server PASSWORD for asynchronous email MDN. |
| AS2_ASYNC_EMAIL_SERVER_FOLDER_NAME| Enter the IMAP folder name for asynchronous email MDN. |
| AS2_HTTP_HOST                  | Required. Enter the server host name. |
| AS2_HTTP_PORT                  | Enter the server HTTP port number. The default is 80. |
| AS2_HTTP_REMOTE_DIRECTORY      | Required. Enter the remote directory name. (Example: /home/) |
**Property** | **Description**
--- | ---
AS2_HTTP_REMOTE_FILENAME | Required. Enter the remote file name.
AS2_HTTP_AUTHTYPE | Enter the HTTP authentication type. Valid values are:
  - AS2_HTTP_AUTHTYPE_NONE - no authentication (Default)
  - AS2_HTTP_AUTHTYPE_BASIC - Use HTTP basic authentication.
  - AS2_HTTP_AUTHTYPE_DIGEST - user HTTP digest authentication.
AS2_HTTP_USERNAME | Enter the username for HTTP authentication.
AS2_HTTP_PASSWORD | Enter the password for HTTP authentication.
AS2_HTTP_ENCTYPE | Set the encryption type. Valid values are:
  - AS2_HTTP_ENCTYPE_NONE - no encryption (default)
  - AS2_HTTP_ENCTYPE_SSL - use secure socket layer (SSL)
AS2_HTTP_TIMEOUT | Enter the time out allowance in milliseconds. Default is 60,000 (1 minute)
AS2_HTTP_PROXY_HOST | Required. Enter the proxy server host name.
AS2_HTTP_PROXY_PORT | Enter the proxy server port number. Default is 80.
AS2_HTTP_PROXY_AUTHTYPE | Enter the HTTP authentication type. Valid values are:
  - AS2_HTTP_AUTHTYPE_NONE - no authentication (Default)
  - AS2_HTTP_AUTHTYPE_BASIC - Use HTTP basic authentication.
  - AS2_HTTP_AUTHTYPE_DIGEST - user HTTP digest authentication.
AS2_HTTP_PROXY_USERNAME | Enter the username for proxy authentication.
AS2_HTTP_PROXY_PASSWORD | Enter the password for HTTP proxy authentication.

**Synchrony**

You can send either synchronous or asynchronous delivery requests to the AS2 servers. By default, the request is synchronous so that you can see the Message Disposition Notification (MDN) immediately in the DeliveryResponse.

If you set the AS2_ASYNC_ADDRESS to your request, the request will be asynchronous. You can specify either an HTTP URL or an email address where the asynchronous MDN will be delivered to. Then, the AS2 server will deliver the MDN directly to that address. Sample code for this is as follows:
Example:

```java
req.addProperty(DeliveryPropertyDefinitions.AS2_ASYNC_ADDRESS, "async_target@acme.com");

req.addProperty(DeliveryPropertyDefinitions.AS2_ASYNC_EMAIL_SERVER_HOST, "mail.acme.com");
req.addProperty(DeliveryPropertyDefinitions.AS2_ASYNC_EMAIL_SERVER_USERNAME, "async_target");
req.addProperty(DeliveryPropertyDefinitions.AS2_ASYNC_EMAIL_SERVER_PASSWORD, "mypassword");

req.setDocument("/document/myEDIdoc");
req.submit(myDeliveryListener);
```

The code above will check the email server periodically after sending the AS2 request to the server. Once the MDN is delivered and found on the email server, the registered DeliveryResponseListener will be called.

## Direct and Buffering Modes

The delivery system supports two modes: Direct mode and Buffering mode. Buffering Mode is the default.

### Direct Mode

Direct Mode offers full, streamlined delivery processing. Documents are delivered to the connection streams that are directly connected to the destinations. This mode is fast, and uses less memory and disk space. It is recommended for online interactive processing.

To set the direct mode, set the BUFFERING_MODE property to “false”. Following is a code sample:
Example
// create delivery manager instance
DeliveryManager dm = new DeliveryManager();

// create a delivery request
DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_IPP_PRINTER);

// set the direct mode
req.addProperty(DeliveryPropertyDefinitions.BUFFERING_MODE, "false");

This mode does not offer document redelivery. For redelivery requirements, use the buffering mode.

Buffering Mode
The buffering mode allows you to redeliver documents as many times as you want. The delivery system uses temporary files to buffer documents, if you specify a temporary directory (ds-temp-dir) in the delivery server configuration file. If you do not specify a temporary directory, the delivery system uses the temporary memory buffer. It is recommended that you define a temporary directory. For more information about the configuration file, see Configuration File Support, page 11-34.

You can explicitly clear the temporary file or buffer by calling DeliveryRequest.close() after finishing your delivery request.

Example
// create delivery manager instance
DeliveryManager dm = new DeliveryManager();

// create a delivery request
DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_IPP_PRINTER);

// set buffering mode
req.addProperty(DeliveryPropertyDefinitions.BUFFERING_MODE, "true");
req.addProperty(DeliveryPropertyDefinitions.TEMP_DIR, "/tmp")

// submit request
req.submit();

// submit request again
req.submit();

// close the request
req.close();
Monitoring Delivery Status

The delivery system allows you to check the latest delivery status of your request by calling the `getStatus()` method. You can check the status of the request anytime, but currently you must retain the delivery request object. Status definitions are defined in the `DeliveryRequest` interface.

Monitoring delivery status is not available for the SMTP and HTTP delivery channels.

Example

```java
// create delivery manager instance
DeliveryManager dm = new DeliveryManager();

// create a delivery request
DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_IPPRINTER);

// submit request
req.submit();

// get request status
int status = req.getStatus();
if (status == DeliveryRequest.STATUS_SUCCESSFUL)
{
    System.out.println("Request has been delivered successfully.
}

// get request status again...
status = req.getStatus();

Global Properties

You can define the global properties to the `DeliveryManager` so that all the delivery requests inherit the global properties automatically.

The following global properties are supported:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUFFERING_MODE</td>
<td>Valid values are &quot;true&quot; (default) and &quot;false&quot;. See Direct and Buffering Modes, page 11-24 for more information.</td>
</tr>
<tr>
<td>TEMP_DIR</td>
<td>Define the location of the temporary directory.</td>
</tr>
<tr>
<td>CA_CERT_FILE</td>
<td>Define the location of the CA Certificate file generated by Oracle Wallet Manager. This is used for SSL connection with the Oracle SSL library. If not specified, the default CA Certificates are used.</td>
</tr>
</tbody>
</table>
Example

// create delivery manager instance
DeliveryManager dm = new DeliveryManager();

    // set global properties
    dm.addProperty(DeliveryPropertyDefinitions.TEMP_DIR, "/tmp");
    dm.addProperty(DeliveryPropertyDefinitions.BUFFERING_MODE, "true");

    // create delivery requests
    DeliveryRequest req1 = dm.createRequest(DeliveryManager.TYPE_IPP_PRINTER);
    DeliveryRequest req2 = dm.createRequest(DeliveryManager.TYPE_IPP_FAX);
    DeliveryRequest req3 = dm.createRequest(DeliveryManager.TYPE_SMTP_EMAIL);
    : :

Delivering Multiple Requests with a Single Output Stream

To deliver your document to multiple delivery channels with a single output stream, use the MultipleRequestHandler utility. Register all your delivery requests to the utility to get a single output stream that internally distributes the data to the requests.
Example

```java
// create delivery manager instance
DeliveryManager dm = new DeliveryManager();

// create delivery requests
DeliveryRequest req1 = dm.createRequest(DeliveryManager.TYPE_IPP_PRINTER);
DeliveryRequest req2 = dm.createRequest(DeliveryManager.TYPE_IPP_FAX);
DeliveryRequest req3 = dm.createRequest(DeliveryManager.TYPE_SMTP_EMAIL);

// create MultipleRequestHandler instance
MultipleRequestHandler mh = new MultipleRequestHandler();
// register delivery requests
mh.addRequest(req1);
mh.addRequest(req2);
mh.addRequest(req3);
// get the ouptput stream
OutputStream out = mh.getDocumentOutputStream();

// write the document
out.write(yourDocument);

// submit all delivery requests
mh.submitRequests();

// close requests
req1.close();
req2.close();
req3.close();
```

Adding a Custom Delivery Channel

You can add custom delivery channels to the system by following the steps below:

1. Define the delivery properties
2. Implement the DeliveryRequest interface
3. Implement the DeliveryRequestHandler interface
4. Implement the DeliveryRequestFactory interface
5. Register your custom DeliveryRequestFactory to the DeliveryManager

The following sections detail how to create a custom delivery channel by creating a sample called "File delivery channel" that delivers documents to the local file system.

Define Delivery Properties

The first step to adding a custom delivery channel is to define the properties. These will vary depending on what you want your channel to do. You can define constants for your
properties. Our example, a file delivery channel requires only one property, which is the destination.

Sample code is:

**Example**

```java
package oracle.apps.xdo.delivery.file;

public interface FilePropertyDefinitions
{
    /** Destination property definition. */
    public static final String FILE_DESTINATION = "FILE_DESTINATION:

    String";

}
```

The value of each constant can be anything, as long as it is a String. It is recommend that you define the value in `[property name]:[property value type]` format so that the delivery system automatically validates the property value at runtime. In the example, the `FILE_DESTINATION` property is defined to have a String value.

**Implement DeliveryRequest Interface**

DeliveryRequest represents a delivery request that includes document information and delivery metadata, such as destination and other properties. To implement `oracle.apps.xdo.delivery.DeliveryRequest` you can extend the class `oracle.apps.xdo.delivery.AbstractDeliveryRequest`.

For example, to create a custom delivery channel to deliver documents to the local file system, the DeliveryRequest implementation will be as follows:

```java
package oracle.apps.xdo.delivery.file;
import oracle.apps.xdo.delivery.AbstractDeliveryRequest;

public class FileDeliveryRequest extends AbstractDeliveryRequest implements FilePropertyDefinitions
{
    private static final String[] MANDATORY_PROPS = {FILE_DESTINATION};

    /**
     * Returns mandatory property names
     */
    public String[] getMandatoryProperties()
    {
        return MANDATORY_PROPS;
    }

    /**
     * Returns optional property names
     */
    public String[] getOptionalProperties()
    {
        return null;
    }
}
```
Implement DeliveryRequestHandler Interface

DeliveryRequestHandler includes the logic for handling the delivery requests. A sample implementation of oracle.apps.xdo.delivery.DeliveryRequestHandler for the file delivery channel is as follows:

Example

```java
package oracle.apps.xdo.delivery.file;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import oracle.apps.xdo.delivery.DeliveryException;
import oracle.apps.xdo.delivery.DeliveryRequest;
import oracle.apps.xdo.delivery.DeliveryRequestHandler;
import oracle.apps.xdo.delivery.DeliveryStatusDefinitions;

public class FileDeliveryRequestHandler implements DeliveryRequestHandler {
    private FileDeliveryRequest mRequest;
    private boolean mIsOpen = false;
    private OutputStream mOut;

    /**
     * default constructor.
     */
    public FileDeliveryRequestHandler() {
    }

    /**
     * sets the request.
     */
    public void setRequest(DeliveryRequest pRequest) {
        mRequest = (FileDeliveryRequest) pRequest;
    }

    /**
     * returns the request.
     */
    public DeliveryRequest getRequest() {
        return mRequest;
    }

    /**
     * opens the output stream to the destination.
     */
    public OutputStream openRequest() throws DeliveryException {
        try {
            String filename =
                (String) mRequest.getProperty(FileDeliveryRequest.FILE_DESTINATION)
            mOut = new BufferedOutputStream(new FileOutputStream(filename));
        } catch (IOException ioe) {
            throw new DeliveryException(ioe.getMessage());
        }
        mIsOpen = true;
        return mOut;
    }

    /**
     * returns the status.
     */
    public DeliveryStatusStatus getStatus() {
        return mRequest.getStatus();
    }

    /**
     * returns the status.
     */
    public DeliveryStatusStatus handleStatus(DeliveryStatusStatus status) {
        mRequest.setStatus(status);
        return status;
    }
}
```

11-30 Oracle XML Publisher User's Guide
mOut = new BufferedOutputStream(new FileOutputStream(filename));

mIsOpen = true;
// set request status to open
mRequest.setStatus(DeliveryStatusDefinitions.STATUS_OPEN);
return mOut;

} catch (IOException e) {
    closeRequest();
    throw new DeliveryException(e);
}

/**
 * flushes and closes the output stream to submit the request.
 */
public void submitRequest() throws DeliveryException {
    try {
        // flush and close
        mOut.flush();
        mOut.close();
        // set request status
        mRequest.setStatus(DeliveryStatusDefinitions.STATUS_SUCCESSFUL);
        mIsOpen = false;
    } catch (IOException e) {
        closeRequest();
        throw new DeliveryException(e);
    }

/**
 * checks the delivery status.
 */
public void updateRequestStatus() throws DeliveryException {
    // check if the file is successfully delivered
    String filename =
        (String) mRequest.getProperty(FileDeliveryRequest.FILE_DESTINATION);
    File f = new File(filename);

    // set request status
    if (f.exists())
        mRequest.setStatus(DeliveryStatusDefinitions.STATUS_SUCCESSFUL);
    else
        mRequest.setStatus(DeliveryStatusDefinitions.STATUS_FAILED_I
O_ERROR);

}  
/**
 * returns the request status.
 */
public boolean isRequestOpen()
{
    return mIsOpen;
}

/**
 * closes the request, frees all resources.
 */
public void closeRequest()
{
    mIsOpen = false;
    try
    {
        if (mOut != null)
        {
            mOut.flush();
            mOut.close();
        }
    }
    catch (IOException e)
    {
    }
    finally
    {
        mOut = null;
    }
}

Implement DeliveryRequestFactory Interface

Implement the DeliveryRequestFactory interface to register your custom delivery channel to the delivery system.

A sample implementation of oracle.apps.xdo.delivery.DeliveryRequestFactory is as follows:
Example
package oracle.apps.xdo.delivery.file;

import oracle.apps.xdo.delivery.DeliveryRequest;
import oracle.apps.xdo.delivery.DeliveryRequestFactory;
import oracle.apps.xdo.delivery.DeliveryRequestHandler;

public class FileDeliveryRequestFactory implements DeliveryRequestFactory {
    /**
     * default constructor.
     */
    public FileDeliveryRequestFactory() {
    }
    /**
     * returns delivery request.
     */
    public DeliveryRequest createRequest() {
        return new FileDeliveryRequest();
    }
    /**
     * returns delivery request handler.
     */
    public DeliveryRequestHandler createRequestHandler() {
        return new FileDeliveryRequestHandler();
    }
    /**
     * returns this
     */
    public DeliveryRequestFactory getFactory() {
        return this;
    }
}

Register your custom DeliveryRequestFactory to DeliveryManager

The final step is to register your custom delivery channel to the delivery system. You can register your delivery channel in two ways:

- **Static method**
  
  Use this method to register your delivery channel to the whole delivery system by specifying it in the configuration file. See Configuration File Support, page 11-34 for more information.

- **Dynamic method**
  
  Register the delivery channel to the Java VM instance by calling the Register API programmatically.
  
  Sample code to register the file delivery channel using the dynamic method and call the file delivery channel is as follows:
Example
package oracle.apps.xdo.delivery.file;

import oracle.apps.xdo.delivery.DeliveryManager;
import oracle.apps.xdo.delivery.DeliveryRequest;

public class FileDeliverySample
{
    public static void main(String[] args) throws Exception
    {
        // register the file delivery channel
        DeliveryManager.addRequestFactory("file", "oracle.apps.xdo.delivery.file.FileDeliveryRequestFactory");

        // create delivery manager instance
        DeliveryManager dm = new DeliveryManager();
        // create a delivery request
        DeliveryRequest req = dm.createRequest("file");

        // set the destination
        req.addProperty(
            FileDeliveryRequest.FILE_DESTINATION,
            "d:/Temp/testDocument_delivered.pdf");
        // set the document to deliver
        req.setDocument("D:/Temp/testDocument.pdf");

        // submit the request
        req.submit();
        // close the request
        req.close();
    }
}

Configuration File Support
The delivery systems supports a configuration file to set default servers, default properties, and custom delivery channels. The location of the configuration file is

{XDO_TOP}/resource/xdodelivery.cfg

where {XDO_TOP} is a Java system property that points to the physical directory.

This system property can be set in two ways:

• Pass -DXDO_TOP=/path/to/xdotop to the Java startup parameter
• Use a Java API in your code, such as java.lang.System.getProperties().put("XDO_TOP", "/path/to/xdotop")

The system property must be defined before constructing a DeliveryManager object.

Following is a sample configuration file:

Example
<?xml version='1.0' encoding='UTF-8'?>
<config xmlns="http://xmlns.oracle.com/oxp/delivery/config">
    <! - ------------------------------
    - >
    <! - servers section
    - >
<! - List your pre-defined servers here. - >

<! - ==================================================== - >

<servers>
  <server name="myprinter1" type="ipp_printer" default="true">
    <uri>ipp://myprinter1.oracle.com:631/printers/myprinter1</uri>
  </server>

  <server name="myprinter2" type="ipp_printer">
    <host>myprinter2.oracle.com</host>
    <port>631</port>
    <uri>ipp://myprinter2.oracle.com:631/printers/myprinter2</uri>
    <authType>basic</authType>
    <username>xdo</username>
    <password>xdo</password>
  </server>

  <server name="myfax1" type="ipp_fax" default="true">
    <host>myfax1.oracle.com</host>
    <port>631</port>
    <uri>ipp://myfax1.oracle.com:631/printers/myfax1</uri>
  </server>

  <server name="mysmtp1" type="smtp_email" default="true">
    <host>myprinter1.oracle.com</host>
    <port>25</port>
  </server>

  <server name="mysmtp2" type="smtp_email">
    <host>mysmtp12.oracle.com</host>
    <port>25</port>
    <username>xdo</username>
    <password>xdo</password>
  </server>
</servers>

<! - properties section - >

<! - List the system properties here. - >

<! - ==================================================== - >

<properties>
  <property name="ds-temp-dir">/tmp</property>
  <property name="ds-buffering">true</property>
</properties>

<! - channels section - >
List the custom delivery channels here.

```
<!-
<!-   channels>

<!-
<!-   channel name="file">oracle.apps.xdo.delivery.file.FileDeliveryRequestFactory</channel>
</channels>
</config>
```

## Defining Multiple Servers for a Delivery Channel

You can define multiple server entries for each delivery channel. For example, the preceding sample configuration file has two server entries for the "ipp_printer" delivery channel ("myprinter1" and "myprinter2").

Load a server entry for a delivery request by calling DeliveryRequest.setServer() method. Following is an example:

### Example

```java
// create delivery manager instance
    DeliveryManager dm = new DeliveryManager();
// create a delivery request
    DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_IPP_PRINTER);

    // load myprinter1 setting
    req.setServer("myprinter1");
```

## Specifying a Default Server for a Delivery Channel

To define a default server for a delivery channel, specify default="true". In the configuration file example above, "myprinter1" is defined as the default server for the "ipp_printer" delivery channel. If a user does not specify the server properties for "ipp_printer" delivery, the server properties under the default server will be used.

## Setting Up CUPS

The delivery manager requires CUPS to print and fax documents. This section describes how to set up CUPS for printing and faxing on RedHat Linux.

The following diagram shows the integration between XML Publisher and CUPS:
The following procedures describe how to add a printer or fax to CUPS and how to test your setup. For more information, see the CUPS Software Administrators Manual (http://www.cups.org/doc-1.1/sam.html) and the Redhat Advanced Server online help.

Prerequisites

- RedHat Advanced Server 3.0
- Fax Modem connected to the Linux box
- Development Tools for the RedHat Advanced Server installed
- CUPS (Installed by default when installing RedHat AS 3.0)

Setting Up a Printer on CUPS

The RedHat Advanced Server provides a configuration wizard to help you set up your printers. The RedHat process is summarized below:

Using the RedHat Printer Configuration Wizard:

1. Run "redhat-config-printer"
   While logged on as the root user, open a terminal and execute "redhat-config-printer". This invokes the Printer configuration window.

2. Select the New tab to launch the Add a new print queue wizard.

3. Follow the wizard prompts to:
   - Enter a queue name.
   - Select the queue type.

   Select "Networked_JetDirect" to set up a network printer. For this selection, you must also enter the following:
   - Printer - enter a hostname or IP address.
   - Port - enter a port.
If the printer driver is installed in Microsoft Windows, the Printer and Port information is available from the Properties dialog for the printer (Settings > Printers and Faxes > (select printer) > File > Properties).

- Select the printer model.

  If your printer supports PostScript, select the following:
  
  - Manufacturer: "Generic"
  - Model: "PostScript Printer"

  • Review your selections and select "Apply" to create your new print queue.

4. Your new queue now displays in the Printer configuration window.

**Test Your Printer on CUPS:**

1. Launch a browser on RedHat and enter the following URL:
   
   http://localhost:631

2. Select the **Printers** tab. The printer you just created will be listed.

   To use your Windows browser to access this page, see Making CUPS Accessible from Other Machines, page 11-40.

3. Select **Print Test Page** to test your printer setup. If the test page does not print, repeat the configuration steps. Ensure that your printer type and model selections are correct.

**Installing and Setting Up Fax for CUPS**

This section describes how to install efax-0.9 software and configure it for CUPS.

**Install the Fax Software:**

1. Download efax-0.9 from one of the following locations:
   
   - http://www.cce.com/efax/download/
   - ftp://ftp.metalab.unc.edu/pub/Linux/apps/serialcomm/fax/efax-0.9.tar.gz

2. Extract the files to a working directory using the following commands:
   
   - gunzip efax-0.9.tar.gz
   - tar xvf efax-0.9.tar

3. Compile and install using the following commands (refer to the Readme for more information):
   
   - make
   - make install

   **Note:** You must have make and gcc installed in your RedHat AS.

4. Test the fax.

   Enter the following command:

   ```
   fax send <fax_number><tiff file>
   ```

   For example:
fax send 1234567 test.tiff

The fax is successful if you get the return code:
done, returning 0 (success)

5. Download fax4CUPS. It is available from the following site:
   • http://www.gnu.org/directory/productivity/special/fax4CUPS.html

6. Install fax4CUPS as follows:
   • Extract the tar file to a temporary directory
   • Change the directory: cd fax4CUPS-1.23
   • Open the INSTALL file and follow all steps.

7. Restart CUPS using the following command:
   /etc/rc.d/init.d/cups restart

Setting Up a Fax on CUPS:
1. Launch a browser and go to the following URL: http://localhost:631/admin
2. Enter the admin username and password in the dialog that launches.
3. From the Admin page, select Add Printer.
4. Add a Fax queue as follows:
   In the Add New Printer region, enter the following fields:
   • Name - enter a meaningful name for the fax, such as "efaxserver". This will be referred to as "ipp://servername:631/printers/efaxserver".
   • Location - optional.
   • Description - optional.
5. Select a device for the fax queue.
   Select "Faxmodem (efax on /dev/modem)". In some cases, "/dev/ttySxx" will be shown instead.
6. Select a model for the fax queue.
   Select "efax". You can also select either "HylaFAX" or "mgetty-fax" if these have been installed.
7. Select the driver for the fax queue.
   Select "efax (en)".
8. Verify that the new fax queue appears on the CUPS Admin Web page.
9. Text the fax on CUPS.
   Enter the following command to test the fax:
   `/usr/bin/lp -d <printer name> -t <phone#> test.pdf
Example:
   `/usr/bin/lp -d efax1 -t 5556231 myfax.pdf`
Making CUPS Accessible from Other Machines

By default, CUPS does not allow access from other network machines. However, it can be configured to allow access, as follows:

1. Open a CUPS configuration file using the following command:
   
   Open /etc/cups/cupsd.conf

2. Add a "Listen" instruction.
   
   • Scroll to the bottom of the configuration file where the other Listen instructions are declared.
   
   • Copy “Listen 127.0.0.1:631” and paste it above or below the original.
   
   • Replace “127.0.0.1” with the Linux server’s IP address.

3. Configure each printer.
   
   • In the configuration file, locate:
     
     <Location /printers/your_printer_queue>
   
   • Comment the instruction "Deny From All".
     
     Example:
     
     # Deny From All
   
   • Change "Allow from 127.0.0.1" to "Allow from All"
   
   • Repeat for all printer or fax queues that you wan to make accessible.

4. Save the configuration file and restart CUPS.
   
   • Use the following command to stop CUPS:
     
     /etc/rc.d/init.d/cups stop
   
   • Use the following command to start CUPS:
     
     /etc/rc.d/init.d/cups start

5. Test the accessibility from other machines.

   Launch a browser from another machine and enter one of the following URLs to ensure that the CUPS web page can be accessed:
   
   • http://linux_server_name:631
   
   • http://linux_ip_address:631
Integrating the Document Viewer into an Application

This chapter covers the following topics:

- Overview
- Parameters
- Implementing the Document Viewer in an Application Page
- Document Viewer Common Region APIs

Overview

The XML Publisher common user interface document viewer, or common region, is an Oracle Applications Framework (OAF) shared region. The document viewer can be run as a standalone page, or it can be integrated within an application flow. The document viewer accepts a set of parameters and renders the output inline, or exports it.

For information on developing applications in Oracle Applications Framework, see the Oracle Applications Framework Developer’s Guide, OracleMetaLink note 269138.1. Specific information about component reuse can be found in Chapter 2: OA Framework Essentials.

Parameters

The viewer region is called with a set of parameters. The values of these parameters will determine how the region will be rendered.
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_DataSource</td>
<td>(Required) The following XML data source types are supported:</td>
</tr>
<tr>
<td></td>
<td>• DATA_SOURCE_TYPE_REQUEST_ID: concurrent program request ID</td>
</tr>
<tr>
<td></td>
<td>• DATA_SOURCE_TYPE_FILE: XML data file</td>
</tr>
<tr>
<td></td>
<td>• DATA_SOURCE_TYPE_BLOB: BlobDomain</td>
</tr>
<tr>
<td></td>
<td>• DATA_SOURCE_TYPE_DOCUMENT: Final document for preview</td>
</tr>
<tr>
<td></td>
<td>These types are defined in the xdo.oa.common.DocumentHelper class.</td>
</tr>
<tr>
<td></td>
<td>Additional parameters may be required depending on the type of data source.</td>
</tr>
<tr>
<td></td>
<td>These parameters are described in the next section.</td>
</tr>
<tr>
<td>p_TemplCode</td>
<td>(Optional) If set to null, the UI will provide a list from which to select</td>
</tr>
<tr>
<td></td>
<td>a template from the available templates based on the p_DataSourceCode parameter. To specify the template to use for this region, enter the Template Code. The template must reside in the Template Manager. Example: AR_CustomerListing</td>
</tr>
<tr>
<td>p_TemplateAppsShortName</td>
<td>Required if p_TemplCode is not null. Enter the Application short name of the application to which the template is assigned in the Template Manager. Example: AR</td>
</tr>
<tr>
<td>p_Locale</td>
<td>(Optional) If null, the UI will provide a list to select available locales</td>
</tr>
<tr>
<td></td>
<td>for the selected template. The value &quot;Default&quot; can be entered to select the default template locale.</td>
</tr>
<tr>
<td>p_OutputType</td>
<td>(Optional) If null, the UI will provide a list to select available output types for the selected template. Valid output types are: RTF, PDF, EXCEL, and HTML, depending on the template type.</td>
</tr>
<tr>
<td>p_XDORegionHeight</td>
<td>Height of the XDO common region window expressed as a percentage. Example: 60%</td>
</tr>
</tbody>
</table>

### Data Source Dependent Parameters

The following parameters are required when the parameter p_DataSource is DocumentHelper.DATA_SOURCE_TYPE_REQUEST_ID. Using this mode the viewer will render a concurrent request output.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_RequestId</td>
<td>Enter the concurrent request ID.</td>
</tr>
</tbody>
</table>

The following parameters are required when the parameter p_DataSource is DocumentHelper.DATA_SOURCE_TYPE_FILE.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_DataSourceCode</td>
<td>Enter the DataSourceCode from the Template Manager repository. Example: AR_CUSTOM_LISTING</td>
</tr>
<tr>
<td>p_DataSourceAppsShortName</td>
<td>Enter the Application Short name for the data source definition. Example: AR</td>
</tr>
<tr>
<td>p.AbsolutePath</td>
<td>Enter the absolute path for the XML data file.</td>
</tr>
</tbody>
</table>

The following parameters are required when the parameter p_DataSource is DocumentHelper.DATA_SOURCE_TYPE_BLOB.
Parameter | Description
---|---
p\_DataSourceCode | Enter the DataSourceCode from the Template Manager repository. Example: AR\_CUSTOM\_LISTING
p\_DataSourceAppsShortName | Enter the Application Short name for the data source definition. Example: AR
XML\_DATA\_FILE | Enter the BLOBDomain that contains the XML data file.

The following parameters are required when the parameter p\_DataSource is DocumentHelper\:DATA\_SOURCE\_TYPE\_DOCUMENT.

Parameter | Description
---|---
p\_AbsolutePath | Enter the absolute path for the document file.
p\_DocumentType | Enter the document type to determine the correct content type. Valid values are: PDF, RTF, EXCEL, HTML.

Implementing the Document Viewer in an Application Page
This section describes the implementation of the common region document viewer within the OA framework application.

Providing Template, Locale, and Format Options
The following figure shows a copy of the Customer Listing Report. The page has two regions: the top is the parameter region, which accepts a set of parameters and the bottom is the output region. The lower region extends the oracle\:apps\:xdo\:oa\:common\:DocumentViewerRn\:xml to render the report output.
Use the Export button to export the output to XML Publisher's supported formats (EXCEL, HTML, PDF, RTF).

Following is the Controller code for the Customer Listing shown in the figure:

```java
public void processRequest(OAPageContext pageContext, OAWebBean webBean) {
    super.processRequest(pageContext, webBean);
    pageContext.putParameter("p_DataSource", DocumentHelper.DATA_SOURCE_TYPE_BLOB);
    pageContext.putParameter("p_DataSourceCode", "CUST_LISTING");
    pageContext.putParameter("p_DataSourceAppsShortName", "XDO");
    pageContext.putParameter("p_XDORegionHeight", "55%");
}
public void processFormRequest(OAPageContext pageContext, OAWebBean webBean) {
    super.processFormRequest(pageContext, webBean);

    OAApplicationModule am = pageContext.getApplicationModule(webBean);

    if (pageContext.getParameter("Go") != null) {
        String customerNameLow = pageContext.getParameter("CustomerNameLow");
        String customerNameHigh = pageContext.getParameter("CustomerNameHigh");
        String customerNoLow = pageContext.getParameter("CustomerNoLow");
        String customerNoHigh = pageContext.getParameter("CustomerNoHigh");
        String orderBy = pageContext.getParameter("OrderBy");

        Serializable[] tcParameter = {customerNameLow, customerNameHigh, customerNoLow, customerNoHigh, orderBy};

        BlobDomain result = (BlobDomain) am.invokeMethod("getXMLData", tcParameter);
        pageContext.putSessionValueDirect("XML_DATA_BLOB", result);
    }
}
```

Using a Fixed Template and Locale Option

In this implementation the region is rendered with a fixed template: "Customer_Listing" and the "Default" locale. Only the Format list is then rendered for the region.

The Controller code for this option is as follows:
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
    super.processRequest(pageContext, webBean);

    pageContext.putParameter("p_DataSource", DocumentHelper.DATA_SOURCE_TYPE_BLOB);
    pageContext.putParameter("p_DataSourceCode", "CUST_LISTING");
    pageContext.putParameter("p_DataSourceAppsShortName", "XDO");
    pageContext.putParameter("p_XDORegionHeight", "55%" );
    pageContext.putParameter("p_TemplateCode", "Customer_Listing");
    pageContext.putParameter("p_TemplateAppsShortName", "XDO");
    pageContext.putParameter("p_Locale", "Default");
}

**Document Viewer Common Region APIs**

The XML Publisher common regions are based on the oracle.apps.xdo.oa.common. DocumentHelper.java API, which has a set of public methods to render and export the document. DocumentHelper.java can also be used outside of the XDO common region.

Following are descriptions of methods to perform the following functions:

- Return the output URL
- Export the document

**DocumentHelper.GetOutputURL**

This method returns the URL, which can then be attached to OAHTML bean to render the document output:

public static String getOutputURL(

    OAPageContext pageContext,
    String appShortName,
    String templateCode,
    InputStream inputStream,
    String outputType,
    Properties properties,
    String language,
    String territory)

A sample implementation of this method is as follows:
String redirectURL = DocumentHelper.getOutputURL(pageContext, appShortName, templateCode, dataInputStream, outputType, properties, language, territory);

OAHTMLWebBean outRegion = (OAHTMLWebBean)createWebBean(pageContext, HTML_WEB_BEAN, null, "IFRAME");
outRegion.setHTMLAttributeValue("src",redirectURL);

outRegion.setHTMLAttributeValue("width", "100");
outRegion.setHTMLAttributeValue("height", "60");
outRegion.setHTMLAttributeValue("title",templateCode);
outRegion.setHTMLAttributeValue("name",templateCode);
pageContext.getPageLayoutBean().addIndexedChild(outRegion);

\textbf{DocumentHelper.exportDocument}

This method can be called from any event, such as the submit button to export the document.

\begin{verbatim}
public static void exportDocument(

    OAPageContext pageContext,
    String appShortName,
    String templateCode,
    String language,
    String territory,
    InputStream inputStream,
    String outputType,
    Properties properties)
\end{verbatim}
XML Publisher Configuration File

XML Publisher Configuration File

Properties set in the Administration interface (see Administration, page 8-1) can also be set using a configuration file. The configuration file is optional. There is no default configuration file in the system.

If you defined properties in the Administration interface and have defined the same properties in a configuration file, the properties defined in the Administration interface will take precedence.

**Important:** It is strongly recommended that you set up a temporary directory for processing large files. If you do not, you will encounter "Out of Memory" errors when processing large files. Create a temporary directory using the Administration interface or by defining the system-temp-dir property (described below).

It is also recommended that you secure the configuration file if you use it to set the PDF security passwords.

When You Must Set a Configuration File

The Administration interface supports the setting of properties at the site level, the data definition level, and the template level. If you have XML Publisher installed on multiple middle tier machines and you want to set a property at the server level, you must use a configuration file.

File Name and Location

You must name this file xdo.cfg and place it under `<XDO_TOP>/resource`. Alternatively, you can place the configuration file under `<JRE_TOP>/jre/lib`.

**Note:** `<JRE_TOP>` refers to AF_JRE_TOP for the concurrent node or OA_JRE_TOP for the Web node.

Namespace

The namespace for this configuration file is:

http://xmlns.oracle.com/oxp/config/

Configuration File Example

Following is a sample configuration file:
<config version="1.0.0" xmlns="http://xmlns.oracle.com/oxp/config/">

  <!-- Properties -->
  <properties>
    <!-- System level properties -->
    <property name="system-temp-dir">/tmp</property>
    <!-- PDF compression -->
    <property name="pdf-compression">true</property>
    <!-- PDF Security -->
    <property name="pdf-security">true</property>
    <property name="pdf-open-password">user</property>
    <property name="pdf-permissions-password">owner</property>
    <property name="pdf-no-printing">true</property>
    <property name="pdf-no-changing-the-document">true</property>
  </properties>

  <!-- Font setting -->
  <fonts>
    <!-- Font setting (for FO to PDF etc...) -->
    <font family="Arial" style="normal" weight="normal">
      <truetype path="/fonts/Arial.ttf" />
    </font>
    <font family="Default" style="normal" weight="normal">
      <truetype path="/fonts/ALBANWTJ.ttf" />
    </font>
    <!-- Font substitute setting (for PDFForm filling etc...) -->
    <font-substitute name="MSGothic">
      <truetype path="/fonts/msgothic.ttc" ttcno="0" />
    </font-substitute>
  </fonts>
</config>

How to Read the Element Specifications

The following is an example of an element specification:

```xml
<Element Name Attribute1="value"
       Attribute2="value"
       AttributeN="value"

   <Subelement Name1>{occurrence-spec}
   <Subelement Name2>...
   <Subelement Name2>
   <Subelement NameN>...
</Element Name>
```

The [occurrence-spec] describes the cardinality of the element, and corresponds to the following set of patterns:

- [0..1] - indicates the element is optional, and may occur only once.
- [0..n] - indicates the element is optional, and may occur multiple times.

Structure

The <config> element is the root element. It has the following structure:
<config version="cdata" xmlns="http://xmlns.oracle.com/oxp/config/">
  <fonts> ... </fonts> [0..n]
  <properties> ... </properties> [0..n]
</config>

Attributes

version
The version number of the configuration file format. Specify 1.0.0.

xmlns
The namespace for XML Publisher’s configuration file. Must be http://xmlns.oracle.com/oxp/config/

Description
The root element of the configuration file. The configuration file consists of two parts:
• Properties (<properties> elements)
• Font definitions (<fonts> elements)
The <fonts> and <properties> elements can appear multiple times. If conflicting definitions are set up, the last occurrence prevails.

Properties
This section describes the <properties> element and the <property> element.

The <properties> element
The properties element is structured as follows:
<properties locales="cdata">
  <property>...
  </property> [0..n]
</properties>

Description
The <properties> element defines a set of properties. You can specify the locales attribute to define locale-specific properties. Following is an example:

Example
<!-- Properties for all locales -->
<properties>
  ...Property definitions here...
</properties>

<!--Korean specific properties-->  
<properties locales="ko-KR">
  ...Korean-specific property definitions here...
</properties>

The <property> element
The <property> element has the following structure:

<property name="cdata">
  ...pcdata...
</property>
Attributes

name
Specify the property name.

Description

Property is a name-value pair. Specify the property name (key) to the name attribute and the value to the element value.

Example

```xml
<properties>
    <property name="system-temp-dir">d:\tmp</property>
    <property name="system-cache-page-size">50</property>
    <property name="pdf-replace-smart-quotes">false</property>
</properties>
```

List of Available Properties

See Configuration, page 8-2 in the Administration chapter for the list of properties. Use the internal name of the property when specifying it in the configuration file. For example, the PDF Output Properties section lists the following:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Internal Name</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compress PDF output</td>
<td>pdf-compression</td>
<td>True</td>
<td>Specify “True” or “False” to control compression of the output PDF file.</td>
</tr>
</tbody>
</table>

To specify this property in the configuration file, enter the following:

```xml
<properties>
    <property name="pdf-compression">false</property>
</properties>
```

Font Definitions

Font definitions include the following elements:

- `<fonts>`
- `<font>`
- `<font-substitute>`
- `<truetype>`
- `<type1>`

For the list of Truetype and Type1 fonts, see Predefined Fonts, page 8-13.

`<fonts>` element

The `<fonts>` element is structured as follows:

```xml
<fonts locales="cdata">
    <font> ... </font> [0..n]
    <font-substitute> ... </font-substitute> [0..n]
</fonts>
```
Attributes

locale
Specify the locales for this font definition. This attribute is optional.

Description

The `<fonts>` element defines a set of fonts. Specify the locales attribute to define locale-specific fonts.

Example

```xml
<!-- Font definitions for all locales -->
<fonts>
  ..Font definitions here...
</fonts>

<!-- Korean-specific font definitions -->
<fonts locales="ko-KR">
  ... Korean Font definitions here...
</fonts>
```

`<font>` element

Following is the structure of the `<font>` element:

```xml
<font family="cdata" style="normalitalic" weight="normalbold">
  <truetype>...</truetype>
  or <type1>...</type1>
</font>
```

Attributes

family
Specify any family name for the font. If you specify "Default" for this attribute, you can define a default fallback font. The family attribute is case-insensitive.

style
Specify "normal" or "italic" for the font style.

weight
Specify "normal" or "bold" for the font weight.

Description

Defines an XML Publisher font. This element is primarily used to define fonts for FO-to-PDF processing (RTF to PDF). The PDF Form Processor (used for PDF templates) does not refer to this element.

Example

```xml
<!-- Define "Arial" font -->
<font family="Arial" style="normal" weight="normal">
  <truetype path="/fonts/arial.ttf"/>
</font>
```

`<font-substitute>` element

Following is the structure of the font-substitute element:
<font-substitute name="cdata">
  <truetype>...</truetype>
or <type1>...</type1>
</font-substitute>

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Specify the name of the font to be substituted.</td>
</tr>
</tbody>
</table>

**Description**

Defines a font substitution. This element is used to define fonts for the PDF Form Processor.

**Example**

```xml
<font-substitute name="MSGothic">
  <truetype path="/fonts/msgothic.ttc" ttccno=0"/>
</font-substitute>
```

### <type1> Element

The form of the `<type1>` element is as follows:

```xml
[type1 name="cdata"/>
```

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Specify one of the Adobe standard Latin1 fonts, such as &quot;Courier&quot;.</td>
</tr>
</tbody>
</table>

**Description**

`<type1>` element defines an Adobe Type1 font.

**Example**

```xml
<!--Define "Helvetica" font as "Serif" -->
<font family="serif" style="normal" weight="normal">
  <type1 name="Helvetica"/>
</font>
```

### Font Fallback Mechanism

When you use the configuration file to define font mappings, you must specify the language-only locale section (for example, "zh") before you specify the language-country locale (for example "zh-TW") sections. The sections must be specified in order because if multiple sections are matched, the last section matched will apply.

The following example shows the proper way to define mappings to achieve the desired fallback for the "zh" and "zh-TW" locales:
With this definition, the following runtime font mappings will occur:

- A template assigned the zh-TW locale using the "xxx" font will result in a mapping to font xxx4zh.ttf.
- A template assigned the zh-TW locale using the "yyy" font will result in a mapping to font yyy4zh-TW.ttf.
- A template assigned the zh or zh-(any other territory) locale will result in a mapping to xxx4zh.ttf or yyy4zh.ttf.
Moving Templates and Data Definitions Between E-Business Suite Instances

Overview

XML Publisher stores the metadata and physical files for templates and data definitions in BLOB columns in its schema. During testing and development you commonly must download information from a development instance to a test instance and then to a production environment. This can be very time consuming, especially if there are many templates to manage. To streamline this process, XML Publisher uses the FNDLOAD loader technology and its own XDOLoader to allow you to move the metadata and physical files for your templates and data definitions from one instance to another.

Use the FNDLOAD utility to upload and download the data definition information and the template metadata stored in the Template Manager.

Use the XDOLoader utility to upload and download the physical files (RTF, PDF, XSL-FO, XML, and XSD).

Using FNDLOAD to Manage Metadata

Because both templates and data definitions are stored in the database, you can use the FNDLOAD loader to download the metadata for these objects and then to upload this metadata to another instance. The FNDLOAD program requires a control file (lct) that XML Publisher provides for use with its objects. The file is called xdotmpl.lct and it is located under your APPL_TOP directory as follows:

XDO_TOP/patch/115/import/xdotmpl.lct

The file structure is as follows:

DO_DS_DEFINITIONS - attributes for data source
| KEY APPLICATION_SHORT_NAME
| KEY DATA_SOURCE_CODE

X_TEMPLATES - attributes for templates
| KEY TMPL_APP_SHORT_NAME
| KEY TEMPLATE_CODE

X_TEMPLATE_FIELDS - template fields
| KEY FIELD_NAME

For more information on the FNDLOAD utility, see Loaders, Oracle Applications System Administrator’s Guide: Configuration.
**Downloading Metadata**

You can use the FNDLOAD utility in conjunction with the xdotmpl.lct control file to download the metadata associated with any single XML Publisher object or the metadata for all data definitions and their associated templates.

The FNDLOAD command takes the following format:

```
FNDLOAD usr/pwd@db 0 Y DOWNLOAD|UPLOAD <full path to xdotmpl.lct>
```

For example, to download all data definitions and templates for Accounts Receivable, use the following command:

```
FNDLOAD apps/apps@mydb 0 Y DOWNLOAD $XDO_TOP/patch/115/import/xdotmpl.lct
```

Where:

- `$XDO_TOP/patch/115/import/xdotmpl.lct` is the full path to the xdotmpl.lct file
- `XMLPData.ldt` is the file the definitions will be downloaded to
- `XDO_DS_DEFINITIONS` introduces the list of XDO attributes that will be used to select the metadata to download
- `APPLICATION_SHORT_NAME=AR` specifies that all XDO metadata for the application short name AR is to be downloaded.

An example to download a single data definition (data source code=ARXCOBLX) and its associated templates is as follows:

```
FNDLOAD user/pwrd@mydb 0 Y DOWNLOAD $XDO_TOP/path/115/import/xdotmpl.lct
```

Where:

- `XMLPData.ldt` introduces the XDO DS_DEFINITIONS APPLICATION_SHORT_NAME=AR DATA_SOURCE_CODE=ARXCOBLX
- An example to download a single data definition (data source code=ARXCOBLX) and a specific template is as follows:

```
FNDLOAD user/pwrd@mydb 0 Y DOWNLOAD $XDO_TOP/path/115/import/xdotmpl.lct
```

Where:

- `XMLPData.ldt` introduces the XDO DS_DEFINITIONS APPLICATION_SHORT_NAME=AR DATA_SOURCE_CODE=ARXCOBLX TEMPLATE_CODE=ARLETTER1
- Note that the data definition code (ARXCOBLX), the template application short name (AR), and the template code (ARLETTER1) are specified.

**Using XDOLoader to Manage Files**

The XDOLoader utility is a Java-based command line program to load template (RTF, PDF, and XSL-FO), XML, and XSD files to the XML Publisher database tables. Use this utility to download files from one instance and load to another.

**Note:** The XDOLoader currently cannot handle XLIFF files.

The XDOLoader has two modes:

- File download only mode
Use this mode to download files from the XDO_LOBS table. Specify the target LOB_CODE, APPS_SHORT_NAME, and LOB_TYPE, LANGUAGE, and TERRITORY to download all files that match the criteria.

- File download and LDT/DRVX generation mode

Use this mode to download files from the XDO_LOBS tables and create and LDT file for the downloaded file.

**Note:** A DRVX file is also created. This file is used by Oracle Development to load templates during patch application. It is not required for use at your site and can be ignored.

Specify the APPS_SHORT_NAME to download all files (including template files, data definition files and sample xml files) that have the same application short name. You can also specify the DS_CODE to select files that are related to the specific data source definition.

**Downloading Files**

To download the files, first set up your environment for your session by setting the APPL_TOP and CLASSPATH. Execute the XDOLoader utility as follows:

```
% java oracle.apps.xdo.ca.util.XDOLoader DOWNLOAD \\
-DB_USERNAME <db_username> \\
-DB_PASSWORD <db_password> \\
-JDBC_CONNECTION <jdbc_con_string> \\
-LOB_TYPE <lob_type> \\
-APPS_SHORT_NAME <application_short_name> \\
-LOB_CODE <lob_code> \\
-LANGUAGE <language> \\
-TERRITORY <territory> \\
-LOG_FILE <log file>
```

The parameters are described in the following table:
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOWNLOAD</td>
<td>(Mandatory) The first parameter: DOWNLOAD will be implemented in the feature.</td>
</tr>
<tr>
<td>DB_USERNAME</td>
<td>(Mandatory) Database user name (example: apps).</td>
</tr>
<tr>
<td>DB_PASSWORD</td>
<td>(Mandatory) Database user password (example: manager).</td>
</tr>
<tr>
<td>JDBC_CONNECTION</td>
<td>(Mandatory) JDBC database connection string (example: ap000sun:1521:db222).</td>
</tr>
<tr>
<td>LOB_TYPE</td>
<td>(Mandatory) XDO LOB type. Valid values are: TEMPLATE XML_SCHEMA XML_SAMPLE</td>
</tr>
<tr>
<td>APPS_SHORT_NAME</td>
<td>(Mandatory) Application short name (example: AR).</td>
</tr>
<tr>
<td>LOB_CODE</td>
<td>(Optional) XDO LOB code. Enter either the Template Code or the Data Definition Code (see below).</td>
</tr>
<tr>
<td>LCT_FILE</td>
<td>(Optional) This is the control file for XML Publisher metadata (see below).</td>
</tr>
<tr>
<td>LANGUAGE</td>
<td>(Mandatory for template files only) ISO two-letter language code (example: en)</td>
</tr>
<tr>
<td>TERRITORY</td>
<td>(Mandatory for template files only) ISO two-letter territory code (example: US)</td>
</tr>
<tr>
<td>LOG_FILE</td>
<td>(Optional) Enter a file name for the uutput log file (default: xdotmpl.log).</td>
</tr>
<tr>
<td>DEBUG</td>
<td>(Optional) Turns debug on or off. Valid values are: true false (default)</td>
</tr>
</tbody>
</table>

The parameters LOB_CODE and LCT_FILE are optional, but one must be defined as follows:

- **LOB_CODE** - use this parameter to download an individual template.
- **LCT_FILE** - if you do not define an LOB_CODE then this parameter is required. If you want to download multiple templates, then you must provide the LCT file. The loaded needs this file to retrieve the templates. The LCT file can be found under $XDO_TOP/patch/115/import/xdotmpl.lct. When you use this option you will not only get the templates, but the ldt file for the templates will be generated for you as well.

**Note:** A DRVX file is also created. This file is used by Oracle Development to load templates during patch application. It is not required for use at your site and can be ignored.
Example
Sample usage is as follows:

```java
java oracle.apps.xdo.oa.util.XDOLoader \
  DOWNLOAD \ 
  -DB_USERNAME apps \ 
  -DB_PASSWORD apps \ 
  -JDBC_CONNECTION ap000sun:1521:apps115 \ 
  -LOB_TYPE TEMPLATE \ 
  -APPS_SHORT_NAME XDO \ 
  -LOB_CODE XDOTMPL1 \ 
  -LANGUAGE ja \ 
  -TERRITORY JP
```

Download usage in LDT/DRVX mode:

```java
java oracle.apps.xdo.oa.util.XDOLoader DOWNLOAD \ 
  -DB_USERNAME <db_username> \ 
  -DB_PASSWORD <db_password> \ 
  -JDBC_CONNECTION <jdbc_conn_string> \ 
  -APPS_SHORT_NAME <application_short_name> \ 
  -DS_CODE (data source code) \ 
  -LCT_FILE <full path to lct file> \ 
  -LDT_FILE <ldt file> \ 
  -DRVX_FILE <drvx file> \ 
  -LOG_FILE <log file>
```

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOWNLOAD</td>
<td>(Mandatory) The first parameter: DOWNLOAD will be implemented in the feature.</td>
</tr>
<tr>
<td>DB_USERNAME</td>
<td>(Mandatory) Database user name (example: apps).</td>
</tr>
<tr>
<td>DB_PASSWORD</td>
<td>(Mandatory) Database user password (example: manager).</td>
</tr>
<tr>
<td>JDBC_CONNECTION</td>
<td>(Mandatory) JDBC database connection string (example: ap000sun:1521:db222).</td>
</tr>
<tr>
<td>APPS_SHORT_NAME</td>
<td>(Mandatory) Application short name (example: AR).</td>
</tr>
<tr>
<td>LCT_FILE</td>
<td>(Mandatory) Full path to the xdotmpl.lct</td>
</tr>
<tr>
<td>DS_CODE</td>
<td>(Optional) Data source code.</td>
</tr>
<tr>
<td>LDT_FILE</td>
<td>(Optional) Output LDT file name (default: xdotmpl.ldt)</td>
</tr>
<tr>
<td>DRVX_FILE</td>
<td>(Optional) Output DRVX file name (default: xdotmpl.drvx)</td>
</tr>
<tr>
<td>LOG_FILE</td>
<td>(Optional) Enter a file name for the output log file (default: xdotmpl.log).</td>
</tr>
<tr>
<td>DEBUG</td>
<td>(Optional) Turns debug on or off. Valid values are: true, false (default)</td>
</tr>
</tbody>
</table>
This mode will create the template or data files, one LDT file, one DRVX file and one log file.

Sample usage is as follows:

Sample usage as follows:
% java oracle.apps.xdo.oa.util.XDOLoader 
DOWNLOAD  
-DB_USERNAME apps \  
-DB_PASSWORD apps \  
-JDBC_CONNECTION ap000sun:1521:apps115 \  
-APPS_SHORT_NAME XDO \  
-LCT_FILE ${XDO_TOP}/patch/115/import/xdotmpl.lct \  
-DS_CODE XDODS1

In this mode the LDT file can be used with the FNDLOAD utility to upload the metadata for the downloaded templates.

**Uploading Files**

To Upload the files, first set up your environment for your session by setting the APPL_TOP and CLASSPATH. Execute the XDOLoader utility as follows:

% java oracle.apps.xdo.oa.util.XDOLoader UPLOAD \  
-DB_USERNAME <db_username> \  
-DB_PASSWORD <db_password> \  
-JDBC_CONNECTION <jdbc_con_string> \  
-LOB_TYPE <lob_type> \  
-APPS_SHORT_NAME <application_short_name> \  
-LOB_CODE <lob_code> \  
-LANGUAGE <language> \  
-TERRITORY <territory> \  
-XDO_FILE_TYPE <xd0_file_type> \  
-NLS_LANG <NLS_LANG> \  
-FILE_CONTENT_TYPE <file_content_type> \  
-FILE_NAME <file_name> \  
-OWNER <owner> \  
-CUSTOM_MODE [FORCE|NOFORCE] \  
-LOG_FILE <log file>

The parameters are described in the following table:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPLOAD</td>
<td>(Mandatory) The first parameter: UPLOAD will be implemented in the feature.</td>
</tr>
<tr>
<td>DB_USERNAME</td>
<td>(Mandatory) Database user name (example: apps).</td>
</tr>
<tr>
<td>DB_PASSWORD</td>
<td>(Mandatory) Database user password (example: manager).</td>
</tr>
<tr>
<td>JDBC_CONNECTION</td>
<td>(Mandatory) JDBC database connection string (example: ap000sun:1521:db222).</td>
</tr>
<tr>
<td>LOB_TYPE</td>
<td>(Mandatory) XDO LOB type. Valid values are:</td>
</tr>
<tr>
<td></td>
<td>TEMPLATE</td>
</tr>
<tr>
<td></td>
<td>XML_SCHEMA</td>
</tr>
<tr>
<td></td>
<td>XML_SAMPLE</td>
</tr>
<tr>
<td>APPS_SHORT_NAME</td>
<td>(Mandatory) Application short name (example: AR).</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LOB_CODE</td>
<td>(Mandatory) XDO LOB code. Enter either the Template Code or the Data Definition Code.</td>
</tr>
<tr>
<td>NLS_LANG</td>
<td>(Mandatory) Enter the NLS_LANG environment variable.</td>
</tr>
<tr>
<td>LANGUAGE</td>
<td>(Optional) ISO two-letter language code (example: en). If NLS_LANGUAGE=&quot;TRADITIONAL CHINESE&quot;, then 'cn_TW' and if NLS_LANGUAGE=&quot;SIMPLIFIED CHINESE&quot; then 'cn_CN' for combination of language and territory.</td>
</tr>
<tr>
<td>TERRITORY</td>
<td>(Mandatory) ISO two-letter territory code (example: US), default is '00'.</td>
</tr>
<tr>
<td>XDO_FILE_TYPE</td>
<td>(Mandatory) Enter the XDO file type, valid values are: PDF, RTF, XLS, XSL-FO, XSL-HTML, XSL-XML, XSL-TEXT, XSD, XML, RTF-ETEXT</td>
</tr>
<tr>
<td>FILE_CONTENT_TYPE</td>
<td>(Optional) Content type of the file (example: text/html, application/pdf)</td>
</tr>
<tr>
<td>FILE_NAME</td>
<td>(Mandatory) Name of the file you want to upload. (example: sample.pdf or test.xml) This file name can be full path (example: /u01/oracle/11iapp/xdo/115/patch/115/publisher/templates)</td>
</tr>
<tr>
<td>OWNER</td>
<td>(Optional) Owner of the template. Default is &quot;ORACLE&quot;.</td>
</tr>
<tr>
<td>CUSTOM_MODE</td>
<td>(Optional) Whether to force update. Valid values are FORCE and NOFORCE (default).</td>
</tr>
<tr>
<td>LOG_FILE</td>
<td>(Optional) Enter a file name for the output log file (default: xdotmpl.log).</td>
</tr>
<tr>
<td>DEBUG</td>
<td>(Optional) Turns debug on or off. Valid values are: true false (default)</td>
</tr>
<tr>
<td>USE_APPS_CONTEXT</td>
<td>(Optional) Whether to use AppsContext or not. Valid values are: true false (default)</td>
</tr>
</tbody>
</table>

If false, '1' is always used for Apps Login ID.

**Example**

Sample usage is as follows:
% java oracle.apps.xdo.oa.util.XDOLoader \
UPLOAD \
-DB_USERNAME apps \
-DB_PASSWORD apps \
-JDBC_CONNECTION ap000sun:1521:apps115 \
-LOB_TYPE TEMPLATE \ 
-APPS_SHORT_NAME XDO \ 
-LOB_CODE XDOTMPL1 \ 
-LANGUAGE ja \ 
-TERRITORY JP \ 
-XDO_FILE_TYPE PDF \ 
-FILE_CONTENT_TYPE 'application/pdf' \ 
-FILE_NAME $XDO_TOP/patch/115/publisher/templates/XDOTMPL1_ja_JP.pdf \ 
-NLS_LANG JAPANESE_JAPAN.JA16EUC

The XDOLoader program can be run either before or after the FNDLOAD command. The files will be loaded with the appropriate LOB_CODE, which will join to the metadata loaded using the TEMPLATE_CODE or DATA_SOURCE_CODE mapping to the LOB_CODE.
Supported XSL-FO Elements

The following table lists the XSL-FO elements supported in this release of XML Publisher. For each element the supported content elements and attributes are listed. If elements have shared supported attributes, these are noted as a group and are listed in the subsequent table, Property Groups. For example, several elements share the content element `inline`. Rather than list the `inline` properties each time, each entry notes that "inline-properties" are supported. The list of inline-properties can then be found in the Property Groups table.

<table>
<thead>
<tr>
<th>Element</th>
<th>Supported Content Elements</th>
<th>Supported Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic-link</td>
<td>external-graphic external-graphic</td>
<td>inline-properties</td>
</tr>
<tr>
<td></td>
<td>inline</td>
<td>external-destination</td>
</tr>
<tr>
<td></td>
<td>leader</td>
<td>internal-destination</td>
</tr>
<tr>
<td></td>
<td>page-number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>page-number-citation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>basic-link</td>
<td></td>
</tr>
<tr>
<td></td>
<td>block</td>
<td></td>
</tr>
<tr>
<td></td>
<td>block-container</td>
<td></td>
</tr>
<tr>
<td></td>
<td>table</td>
<td></td>
</tr>
<tr>
<td></td>
<td>list-block</td>
<td></td>
</tr>
<tr>
<td></td>
<td>wrapper</td>
<td></td>
</tr>
<tr>
<td>bidi-override</td>
<td>bidi-override external-graphic</td>
<td>inline-properties</td>
</tr>
<tr>
<td></td>
<td>instream-foreign-object</td>
<td></td>
</tr>
<tr>
<td></td>
<td>inline</td>
<td></td>
</tr>
<tr>
<td></td>
<td>leader</td>
<td></td>
</tr>
<tr>
<td></td>
<td>page-number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>page-number-citation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>basic-link-citation</td>
<td></td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Element</th>
<th>Supported Content Elements</th>
<th>Supported Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>block</td>
<td>external-graphic</td>
<td>block-properties</td>
</tr>
<tr>
<td></td>
<td>inline</td>
<td></td>
</tr>
<tr>
<td></td>
<td>page-number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>page-number-citation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>basic-link</td>
<td></td>
</tr>
<tr>
<td></td>
<td>block</td>
<td></td>
</tr>
<tr>
<td></td>
<td>block-container</td>
<td></td>
</tr>
<tr>
<td></td>
<td>table</td>
<td></td>
</tr>
<tr>
<td></td>
<td>list-block</td>
<td></td>
</tr>
<tr>
<td></td>
<td>wrapper</td>
<td></td>
</tr>
<tr>
<td>block-container</td>
<td>block</td>
<td>block-properties</td>
</tr>
<tr>
<td></td>
<td>block-container</td>
<td></td>
</tr>
<tr>
<td></td>
<td>table</td>
<td></td>
</tr>
<tr>
<td></td>
<td>list-block</td>
<td></td>
</tr>
<tr>
<td></td>
<td>wrapper</td>
<td></td>
</tr>
<tr>
<td>conditional-page-master-</td>
<td>N/A</td>
<td>master-reference</td>
</tr>
<tr>
<td>reference</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>page-position</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• first</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• rest</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• any</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• inherit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>odd-or-even</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• odd</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• even</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• any</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• inherit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>blank-or-not-blank</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• blank</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• not-blank</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• any</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• inherit</td>
<td></td>
</tr>
<tr>
<td>external-graphic</td>
<td>N/A</td>
<td>graphic-properties</td>
</tr>
<tr>
<td></td>
<td></td>
<td>src</td>
</tr>
</tbody>
</table>

Oracle XML Publisher User’s Guide
<table>
<thead>
<tr>
<th>Element</th>
<th>Supported Content Elements</th>
<th>Supported Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>flow</td>
<td>block, block-container, table, list-block, wrapper</td>
<td>flow-properties</td>
</tr>
<tr>
<td>inline</td>
<td>external-graphic, inline, leader, page-number, page-number-citation, basic-link, block, block-container, table, wrapper</td>
<td>inline-properties</td>
</tr>
<tr>
<td>instream-foreign-object</td>
<td>N/A</td>
<td>graphic-properties</td>
</tr>
<tr>
<td>layout-master-set</td>
<td>page-sequence-master, simple-page-master, simple-page-master, page-sequence-master</td>
<td>N/A</td>
</tr>
<tr>
<td>leader</td>
<td>N/A</td>
<td>inline-properties</td>
</tr>
<tr>
<td>list-block</td>
<td>list-item</td>
<td>block-properties</td>
</tr>
<tr>
<td>list-item</td>
<td>list-item-label, list-item-body</td>
<td>block-properties</td>
</tr>
<tr>
<td>list-item-body</td>
<td>block, block-container, table, list-block, wrapper</td>
<td>block-properties</td>
</tr>
<tr>
<td>list-item-label</td>
<td>block, block-container, table, list-block, wrapper</td>
<td>block-properties</td>
</tr>
<tr>
<td>page-number</td>
<td>N/A</td>
<td>empty-inline-properties</td>
</tr>
<tr>
<td>page-number-citation</td>
<td>N/A</td>
<td>empty-inline-properties ref-id</td>
</tr>
<tr>
<td>Element</td>
<td>Supported Content Elements</td>
<td>Supported Attributes</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------------------------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>page-sequence</td>
<td>static-content</td>
<td>inheritable-properties</td>
</tr>
<tr>
<td></td>
<td>flow</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>master-reference</td>
</tr>
<tr>
<td></td>
<td></td>
<td>initial-page-number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>format</td>
</tr>
<tr>
<td>page-sequence-master</td>
<td>single-page-master-reference</td>
<td>master-name</td>
</tr>
<tr>
<td></td>
<td>repeatable-page-master-reference</td>
<td></td>
</tr>
<tr>
<td></td>
<td>repeatable-page-master-alternatives</td>
<td></td>
</tr>
<tr>
<td>region-after</td>
<td>N/A</td>
<td>side-region-properties</td>
</tr>
<tr>
<td>region-before</td>
<td>N/A</td>
<td>side-region-properties</td>
</tr>
<tr>
<td>region-body</td>
<td>N/A</td>
<td>region-properties</td>
</tr>
<tr>
<td></td>
<td></td>
<td>margin-properties-CSS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>column-count</td>
</tr>
<tr>
<td>region-end</td>
<td>N/A</td>
<td>side-region-properties</td>
</tr>
<tr>
<td>region-start</td>
<td>N/A</td>
<td>side-region-properties</td>
</tr>
<tr>
<td>repeatable-page-master-</td>
<td>conditional-page-master-reference</td>
<td>maximum-repeats</td>
</tr>
<tr>
<td>alternatives</td>
<td></td>
<td></td>
</tr>
<tr>
<td>repeatable-page-master-</td>
<td>N/A</td>
<td>master-reference maximum-repeats</td>
</tr>
<tr>
<td>reference</td>
<td></td>
<td></td>
</tr>
<tr>
<td>root</td>
<td>layout-master-set</td>
<td>inheritable-properties</td>
</tr>
<tr>
<td></td>
<td>page-sequence</td>
<td></td>
</tr>
<tr>
<td>simple-page-master</td>
<td>region-body</td>
<td>margin-properties-CSS</td>
</tr>
<tr>
<td></td>
<td>region-before</td>
<td>master-name</td>
</tr>
<tr>
<td></td>
<td>region-after</td>
<td>page-height</td>
</tr>
<tr>
<td></td>
<td>region-start</td>
<td>page-width</td>
</tr>
<tr>
<td></td>
<td>region-end</td>
<td>writing-mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>lr-tb</td>
</tr>
<tr>
<td>single-page-master-reference</td>
<td>N/A</td>
<td>master-reference</td>
</tr>
<tr>
<td>static-content</td>
<td>block</td>
<td>flow-properties</td>
</tr>
<tr>
<td></td>
<td>block-container</td>
<td></td>
</tr>
<tr>
<td></td>
<td>table</td>
<td></td>
</tr>
<tr>
<td></td>
<td>wrapper</td>
<td></td>
</tr>
<tr>
<td>table</td>
<td>table-column</td>
<td>block-properties</td>
</tr>
<tr>
<td></td>
<td>table-header</td>
<td></td>
</tr>
<tr>
<td></td>
<td>table-footer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>table-body</td>
<td></td>
</tr>
<tr>
<td>Element</td>
<td>Supported Content Elements</td>
<td>Supported Attributes</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>table-body</td>
<td>table-row</td>
<td>inheritable-properties id</td>
</tr>
<tr>
<td></td>
<td>block</td>
<td>block-properties</td>
</tr>
<tr>
<td></td>
<td>block-container</td>
<td>number-columns-spanned</td>
</tr>
<tr>
<td></td>
<td>table</td>
<td>number-rows-spanned</td>
</tr>
<tr>
<td></td>
<td>list-block</td>
<td></td>
</tr>
<tr>
<td></td>
<td>wrapper</td>
<td></td>
</tr>
<tr>
<td>table-column</td>
<td>N/A</td>
<td>inheritable-properties column-number column-width number-columns-repeated</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>table-footer</td>
<td>table-row</td>
<td>inheritable-properties id</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>table-header</td>
<td>table-row</td>
<td>inheritable-properties id</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>table-row</td>
<td>table-cell</td>
<td>inheritable-properties id</td>
</tr>
<tr>
<td>wrapper</td>
<td>inline</td>
<td>inheritable-properties id</td>
</tr>
<tr>
<td></td>
<td>page-number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>page-number-citation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>basic-link</td>
<td></td>
</tr>
<tr>
<td></td>
<td>block</td>
<td></td>
</tr>
<tr>
<td></td>
<td>block-container</td>
<td></td>
</tr>
<tr>
<td></td>
<td>table</td>
<td></td>
</tr>
<tr>
<td></td>
<td>wrapper</td>
<td></td>
</tr>
</tbody>
</table>

**Property Groups Table**

The following table lists the supported properties belonging to the attribute groups defined in the preceding table.

<table>
<thead>
<tr>
<th>Property Group</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>area-properties</td>
<td>overflow (visible, hidden)</td>
</tr>
<tr>
<td></td>
<td>writing-mode (lr-tb, rl-tb, lr, rl)</td>
</tr>
<tr>
<td></td>
<td>baseline-shift (baseline, sub, super)</td>
</tr>
<tr>
<td></td>
<td>vertical-align</td>
</tr>
<tr>
<td>block-properties</td>
<td>inheritable-properties</td>
</tr>
<tr>
<td></td>
<td>id</td>
</tr>
<tr>
<td>Property Group</td>
<td>Properties</td>
</tr>
<tr>
<td>----------------------------------------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>border-padding-background-properties</td>
<td>background-color</td>
</tr>
<tr>
<td></td>
<td>background-image</td>
</tr>
<tr>
<td></td>
<td>background-position-vertical</td>
</tr>
<tr>
<td></td>
<td>background-position-horizontal</td>
</tr>
<tr>
<td></td>
<td>border</td>
</tr>
<tr>
<td></td>
<td>border-after-color</td>
</tr>
<tr>
<td></td>
<td>border-after-style (none, solid)</td>
</tr>
<tr>
<td></td>
<td>border-after-width</td>
</tr>
<tr>
<td></td>
<td>border-before-color</td>
</tr>
<tr>
<td></td>
<td>border-before-style (none, solid)</td>
</tr>
<tr>
<td></td>
<td>border-before-width</td>
</tr>
<tr>
<td></td>
<td>border-bottom</td>
</tr>
<tr>
<td></td>
<td>border-bottom-color</td>
</tr>
<tr>
<td></td>
<td>border-bottom-style (none, solid)</td>
</tr>
<tr>
<td></td>
<td>border-bottom-width</td>
</tr>
<tr>
<td></td>
<td>border-color</td>
</tr>
<tr>
<td></td>
<td>border-end-color</td>
</tr>
<tr>
<td></td>
<td>border-end-style (none, solid)</td>
</tr>
<tr>
<td></td>
<td>border-end-width</td>
</tr>
<tr>
<td></td>
<td>border-left</td>
</tr>
<tr>
<td></td>
<td>border-left-color</td>
</tr>
<tr>
<td></td>
<td>border-left-style (none, solid)</td>
</tr>
<tr>
<td></td>
<td>border-left-width</td>
</tr>
<tr>
<td></td>
<td>border-right</td>
</tr>
<tr>
<td></td>
<td>border-right-color</td>
</tr>
<tr>
<td></td>
<td>border-right-style (none, solid)</td>
</tr>
<tr>
<td></td>
<td>border-right-width</td>
</tr>
<tr>
<td></td>
<td>border-start-color</td>
</tr>
<tr>
<td></td>
<td>border-start-style (none, solid)</td>
</tr>
<tr>
<td></td>
<td>border-start-width</td>
</tr>
<tr>
<td></td>
<td>border-top</td>
</tr>
<tr>
<td></td>
<td>border-top-color</td>
</tr>
<tr>
<td></td>
<td>border-top-style (none, solid)</td>
</tr>
<tr>
<td></td>
<td>border-top-width</td>
</tr>
<tr>
<td></td>
<td>border-width</td>
</tr>
<tr>
<td></td>
<td>padding</td>
</tr>
<tr>
<td></td>
<td>padding-after</td>
</tr>
<tr>
<td></td>
<td>padding-before</td>
</tr>
<tr>
<td></td>
<td>padding-bottom</td>
</tr>
<tr>
<td></td>
<td>padding-end</td>
</tr>
<tr>
<td></td>
<td>padding-left</td>
</tr>
<tr>
<td></td>
<td>padding-right</td>
</tr>
<tr>
<td></td>
<td>padding-start</td>
</tr>
<tr>
<td></td>
<td>padding-top</td>
</tr>
<tr>
<td>Property Group</td>
<td>Properties</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>box-size-properties</td>
<td>height, width</td>
</tr>
<tr>
<td>character-properties</td>
<td>font-properties, text-decoration</td>
</tr>
<tr>
<td>empty-inline-properties</td>
<td>character-properties, border-padding-background-properties, id, color</td>
</tr>
<tr>
<td>flow-properties</td>
<td>inheritable-properties, id, flow-name</td>
</tr>
<tr>
<td>font-properties</td>
<td>font-family, font-size, font-style (normal, italic, oblique), font-weight (normal, bold), table-omit-header-at-break (TRUE, FALSE, inherit), table-omit-footer-at-break (TRUE, FALSE, inherit)</td>
</tr>
<tr>
<td>graphic-properties</td>
<td>border-padding-background-properties, margin-properties-inline, box-size-properties, font-properties, keeps-and-breaks-properties-atomic, id</td>
</tr>
<tr>
<td>inheritable-properties</td>
<td>border-padding-background-properties, box-size-properties margin-properties-inline, area-properties, character-properties, line-related-properties, leader-properties, keeps-and-breaks-properties-block color</td>
</tr>
<tr>
<td>inline-properties</td>
<td>inheritable-properties, id</td>
</tr>
<tr>
<td>keeps-and-breaks-properties-atomic</td>
<td>break-after (auto, column, page), break-before (auto, column)</td>
</tr>
<tr>
<td>keeps-and-breaks-properties-block</td>
<td>keeps-and-breaks-properties-inline</td>
</tr>
<tr>
<td>Property Group</td>
<td>Properties</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| keeps-and-breaks-properties-inline| keeps-and-breaks-properties-atomic  
keep-together  
keep-together.within-line  
keep-together.within-column  
keep-together.within-page |
| leader-properties                | leader-pattern (rule, dots)  
leader-length  
leader-length.optimum (solid)  
rule-thickness |
| line-related-properties          | text-align (start, center, end, justify, right, inherit)  
text-align-last (start, center, end, justify, left, right, inherit)  
text-indent  
linefeed-treatment (ignore, preserve, treat-as-space, treat-as-zero-width-space, inherit )  
white-space-treatment (ignore, preserve, ignore-if-before-linefeed, ignore-if-after-linefeed, ignore-if-surrounding-linefeed, inherit)  
white-space-collapse (FALSE, TRUE, inherit)  
wrap-option (no-wrap, wrap, inherit)  
direction (ltr) |
| margin-properties-block          | margin-properties-CSS  
space-after  
space-after.optimum  
space-before  
space-before.optimum  
start-indent  
end-indent |
| margin-properties-CSS            | margin  
margin-bottom  
margin-left  
margin-right  
margin-top |
| margin-properties-inline         | margin-properties-block  
space-start  
space-start.optimum  
space-end  
space-end.optimum |
<table>
<thead>
<tr>
<th>Property Group</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>region-properties</td>
<td>border-padding-background-properties</td>
</tr>
<tr>
<td></td>
<td>area-properties</td>
</tr>
<tr>
<td></td>
<td>region-name</td>
</tr>
<tr>
<td>side-region-properties</td>
<td>region-properties</td>
</tr>
<tr>
<td></td>
<td>extent</td>
</tr>
</tbody>
</table>
Index

A
administration
  configuration tab, 8-2
alignment
  RTF template, 2-38
APIs, 10-1

B
background support
  RTF templates, 2-42
bidirectional language alignment
  RTF template, 2-38
body tags
  PDF template, 3-8
  RTF template, 2-13
bookmarks
  generating PDF bookmarks from an RTF template, 2-51
  inserting in RTF templates, 2-48
brought forward/carried forward page totals, 2-67
buffering mode
  delivery server, 11-24
  bursting engine, 10-43

C
calculations in PDF template, 3-13
calendar profile option, 2-102
calendar specification, 2-102
cell highlighting
  conditional in RTF templates, 2-62
charts
  building in RTF templates, 2-15
check box placeholder
  creating in PDF template, 3-6
check box support
  RTF templates, 2-51
choose statements, 2-57
clip art support, 2-26
columns
  fixed width in tables, 2-39
common region, 12-1
  APIs, 12-5
concurrent manager, 6-1, 6-1
concurrent program

for data template, 9-20
conditional columns
  rtf template, 2-59
conditional formatting, 2-55
  table rows, 2-61
conditional formatting features, 2-55
configuration
  data definition level, 5-3
  template level, 5-9
configuration file, A-1
  <properties> element, A-3
  <root> element, A-2
delivery manager, 11-34
structure, A-2
configuration properties
  precedence of levels, 8-2
configuration tab
  administration, 8-2
copy command, 2-107
Create Template page, 5-6
Create Data Definition page, 5-2
definitions, 5-2
Create Template page
  field definitions, 5-4
cross-tab reports, 2-85
CUPS setup, 11-36
currencies
  defining format sets, 8-16

D
Data Definition
  creating in Template Manager, 5-2
data definition properties
  setting, 5-3
data template
  calling, 9-20
  constructing, 9-5
  setting up concurrent program for, 9-20
data template definition, 9-2
date fields in RTF templates, 2-40
date formatting
  RTF templates, 2-91
default template, 5-5
default template file, 5-4
delivery channels
adding custom, 11-28
delivery manager
configuration file, 11-34
delivery server
buffering mode, 11-24
direct mode, 11-24
global properties, 11-26
multiple requests, 11-27
delivery status, 11-26
direct mode
delivery server, 11-24
document viewer, 12-1
APIs, 12-5
implementing, 12-3
parameters, 12-1
downloaded PDFs as templates, 3-15
downloading objects, B-1
drawing support, 2-26
drop-down form field support
RTF templates, 2-53
dynamic data columns, 2-88
example, 2-89
dynamic table of contents in RTF template, 2-51

E
e-mail delivery, 11-2
editing templates from the Template Manager, 5-7
enabling a translation, 5-12
etext data tables, 4-5
etext template command rows, 4-5
etext template setup command table, 4-12

F
fax delivery, 11-9
fixed-width columns
RTF templates, 2-39
FO
supported elements, C-1
FO elements
using in RTF templates, 2-112, 7-5
font definitions
configuration file, A-4
font fallback
configuration file, A-6
font fallback mechanism, 8-12
font files
administration, 8-10
location, 8-13
uploading, 8-10
font mapping
administration, 8-10
fonts
external, 2-103
setting up, 2-103
footers
RTF template, 2-13
for-each-group XSL 2.0 standard, 2-72
form field method
inserting placeholders, 2-7
form field properties options in PDF template, 3-6
form fields in the PDF template, 3-4
formatting options in PDF templates, 3-6
FTP delivery, 11-13

G
generating output, 6-1
global properties
delivery server, 11-26
groups
basic RTF method, 2-11
defining in PDF template, 3-8
defining in RTF template, 2-10
syntax, 2-10
defining in RTF templates, 2-5
form field method, 2-11
grouping scenarios in RTF template, 2-10
in RTF templates, 2-5

H
headers and footers
different first page, 2-14
different odd and even pages, 2-14
inserting placeholders, 2-13
multiple, 2-13
resetting within one output file, 2-83
RTF template, 2-13
horizontal table break, 2-88
HTTP
delivering documents over, 11-18
hyperlinks
bookmarks, 2-48
dynamic, 2-48
inserting in RTF template, 2-48
internal, 2-48
static, 2-48

I
IF statements
in free-form text, 2-56
if statements, 2-55, 2-55
if-then-else statements, 2-57
images
including in RTF template, 2-14
importing templates, 2-113
incomplete translations, 5-12

L
languages
adding templates for, 5-8
last page
support for special content, 2-46
locales
configuration file, 8-12

M

markup
  adding to the PDF template, 3-4
  adding to the RTF template, 2-6
merging PDF files, 10-28
multicolumn page support, 2-41
multiple headers and footers
  RTF template, 2-13

N
Namespace support in RTF template, 2-107
native page breaks and page numbering, 2-37
nulls
  how to test for in XML data, 2-71
number formatting
  RTF templates, 2-91

O
Out of memory error
  avoiding, 8-3
overflow data in PDF templates, 3-15

P
page breaks
  PDF templates, 3-9
  RTF template, 2-37, 2-44
page breaks and page numbering
  native support, 2-37
page number
  setting initial
    RTF templates, 2-45
page numbers
  PDF templates, 3-9
  restarting within one output file, 2-83
  RTF template, 2-37
page totals
  brought forward/carried forward, 2-67
  inserting in RTF template, 2-64
parameters
  defining in RTF template, 2-79
PDF files
  merging, 10-28
PDF template
  template mapping, 5-4
PDF template mapping, 5-5, 5-8
PDF templates
  adding markup, 3-4
  completed example, 3-14
  defining groups, 3-8
  definition of, 3-1
downloading PDFs to use as templates, 3-15
overflow data, 3-15
page breaks, 3-9
page numbering, 3-9
placeholders
  check box, 3-6
  naming, 3-5
  naming to avoid mapping, 3-5
  radio button group, 3-7
text, 3-5
types of, 3-5
placement of repeating fields at runtime, 3-15
runtime behaviors, 3-15
sample purchase order template, 3-2
saving as Adobe Acrobat 5.0 compatible, 3-2
sources for document templates, 3-2
supported modes, 3-2
when to use, 3-1
placeholders
  basic RTF method, 2-7, 2-7
  form field RTF method, 2-7, 2-7
  in PDF templates, 3-4
  in RTF templates, 2-5
  defining, 2-5, 2-6
  inserting in the header and footer of RTF template, 2-13
PDF templates
  check box, 3-6
  naming, 3-5
  naming to avoid mapping, 3-5
  radio button group, 3-7
text, 3-5
types of, 3-5
predefined fonts, 8-13
previewing a template, 5-7
printing, 11-6
process overview, 1-3
  design time, 1-3
  runtime, 1-4
progress indicator
  translations, 5-12
properties
  setting at template level, 2-81
properties element
  configuration file, A-3
publishing, 6-1
radio button group
  creating in PDF templates, 3-7
regrouping, 2-72
repeating elements
  See groups
Rich Text Format (RTF)
  definition, 2-1
row breaking
  preventing in RTF templates, 2-39

Index-3
row formatting
  conditional, 2-61
RTF placeholders
  syntax, 2-6
RTF template
  adding markup, 2-6
  applying design elements, 2-6
  definition, 2-1
  designing, 2-2
  groups, 2-5
  including images, 2-14
  native formatting features, 2-37
  placeholders, 2-5
  prerequisites, 2-2
  sample template design, 2-3
  supported modes, 2-2
    basic method, 2-2
    form field method, 2-2
    using XSL or XSL:FO, 2-2
RTF template design
  headers and footers, 2-13
RTF template placeholders, 2-6
running totals
  RTF templates, 2-70
runtime data definition properties
S
sample RTF template
  completed markup, 2-9
section context command, 2-83
secure ftp
  delivery, 11-14
seeded templates
  deleting, 5-7
  modifying, 5-7
setRowsetTag method, 9-24
setRowsTag method, 9-24
setSQL method, 9-24
setting the initial page number
  RTF templates, 2-45
shape support, 2-26
sorting
  RTF template, 2-71
SQL functions
  using in RTF templates, 2-104
XML Publisher syntax for, 7-1
SQL functions extended for XML Publisher, 7-1
Standard Request Submission, 6-1
status indicator
  translations, 5-12
subtemplates
  defining in your template, 2-113
  syntax
    RTF template placeholder, 2-6
T
  table features
    fixed-width columns, 2-39
    preventing rows breaking across pages
      RTF template, 2-39
    text truncation, 2-39
  table features
    repeating table headers
      RTF template, 2-38
      RTF template, 2-38
  table of contents support
    RTF template, 2-51
      dynamic TOC, 2-51
  tables
    best practices for formatting, 2-112
  tables
    horizontal table break, 2-88
  template
    defining in Template Manager, 5-4
  Template Builder, 2-3
  Template Manager
    accessing, 5-1
    copying a template, 5-6
    creating a Data Definition, 5-2
    defining the template, 5-4
    description, 1-2
    editing templates, 5-7
    previewing a template, 5-7
    template mapping, 5-8
    updating a template, 5-6
    viewing a template, 5-6
  Template Manager
    features, 5-1
  template mapping, 5-4, 5-5
  template properties
    setting, 5-9
  templates
    copying, 5-6
    editing in the Template Manager, 5-7
    updating, 5-7
  temporary directory
    setting, 8-3
text placeholder
  creating in PDF template, 3-5
text truncation in tables, 2-39
totals
  brought forward/carried forward, 2-67
  inserting page totals in RTF template, 2-64
  running
    RTF templates, 2-70
  translatable templates, 5-9
translations
  enabling, 5-12
  incomplete, 5-12
  progress indicator, 5-12
  status indicator, 5-12

Index-4
Update Configuration button
  data definition, 5-3
templates, 5-9
updateable variables
  RTF templates, 2-79
updating a data definition, 5-3
updating a template, 5-6
uploading objects, B-1
user interfaces, 1-2

variables
  RTF templates, 2-79
viewing a data definition, 5-3
viewing a template, 5-6

watermarks
  RTF templates, 2-42
WebDAV delivery, 11-10

XDOloader, B-1
XLIFF file, 5-10
XML data file
  example, 2-4
XML file
  how to read, 2-4
XML Report Publisher concurrent request
description, 1-2
XML Report Publisher program, 6-1
XPath Support in RTF Templates, 2-104
XSL elements
  apply a template rule, 2-110
call template, 2-111
copy the current node, 2-110
define the root element of the stylesheet, 2-111
import stylesheet, 2-111
template declaration, 2-111
using in RTF templates, 2-110
variable declaration, 2-111
XML Publisher syntax for, 7-4
XSL:FO elements
  using in RTF templates, 2-104