

**Oracle® XML Publisher**

Administration and Developer's Guide

Release 11*i*

**Part No. E05321-01**

April 2007

Oracle XML Publisher Administration and Developer's Guide, Release 11i

Part No. E05321-01

Copyright © 2006, 2007, Oracle. All rights reserved.

Primary Author: Leslie G. Studdard

Contributing Author: Ahmed Ali, Tim Dexter, Klaus Fabian, Incheol Kang, Kazuko Kawahara, Kei Saito, Ashish Shrivastava, Jackie Yeung

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**

## **1 Introduction**

What is XML Publisher?.....	1-1
Structure of the XML Publisher Documentation Set.....	1-2

## **2 Using the Template Manager**

<b>Introduction</b> .....	2-1
<b>Creating the Data Definition</b> .....	2-2
Viewing and Updating a Data Definition.....	2-4
<b>Creating the Template</b> .....	2-5
Copying a Template.....	2-8
<b>Viewing and Updating a Template</b> .....	2-8
Updating the Template General Definitions.....	2-10
Previewing a Template.....	2-10
Editing the Template Layout.....	2-10
Adding Localized Templates for Additional Languages .....	2-10
Mapping PDF Template Fields.....	2-11
Setting Runtime Properties for a Template.....	2-12
<b>Translatable Templates</b> .....	2-12

## **3 Generating Your Customized Report**

Using the Concurrent Manager to Generate Your Custom Output.....	3-1
------------------------------------------------------------------	-----

## 4 Administration

XML Publisher Administration.....	4-1
Setting Runtime Properties.....	4-2
Uploading Font Files.....	4-13
Creating Font Mappings.....	4-13
Locales.....	4-15
Font Fallback Logic.....	4-16
Font File Location.....	4-17
Predefined Fonts.....	4-17
Defining Currency Formats.....	4-21

## 5 Data Templates

Introduction.....	5-1
The Data Template Definition.....	5-3
Constructing the Data Template.....	5-7
How to Call a Data Template.....	5-27
Calling a Data Template from the Concurrent Manager.....	5-27
Calling a Data Template from the Java API.....	5-29
Distributed Queries.....	5-32
Sample Data Templates.....	5-34

## 6 Calling XML Publisher APIs

Introduction.....	6-1
XML Publisher Core APIs.....	6-2
PDF Form Processing Engine.....	6-3
RTF Processor Engine.....	6-8
FO Processor Engine.....	6-10
PDF Document Merger.....	6-21
PDF Book Binder Processor.....	6-28
Document Processor Engine.....	6-31
Bursting Engine.....	6-44
XML Publisher Properties.....	6-55
Applications Layer APIs.....	6-59
Datasource APIs.....	6-60
Template APIs.....	6-62
Advanced Barcode Font Formatting Implementation.....	6-70

## 7 Using the Delivery Manager APIs

Introduction.....	7-1
Delivering Documents via e-Mail.....	7-2
Delivering Your Document to a Printer.....	7-8
Delivering Your Documents via Fax.....	7-14
Delivering Your Documents to WebDAV Servers.....	7-15
Deliver Your Documents Using FTP.....	7-17
Delivering Documents over Secure FTP.....	7-19
Delivering Documents over HTTP.....	7-22
Delivering Documents via AS2.....	7-24
Delivering Documents Using an External Command.....	7-31
Delivering Documents to the Local File System.....	7-32
Direct and Buffering Modes.....	7-33
Asynchronous Delivery Requests.....	7-34
Document Filter Support.....	7-35
Date Expression Support.....	7-36
Internationalization.....	7-36
Monitoring Delivery Status.....	7-37
Global Properties.....	7-38
Adding a Custom Delivery Channel.....	7-39
Configuration File Support.....	7-45
Setting Up CUPS.....	7-49

## 8 Integrating the Document Viewer into an Application

Overview.....	8-1
Parameters.....	8-1
Implementing the Document Viewer in an Application Page.....	8-4
Document Viewer Common Region APIs.....	8-6

## A Setting Up XML Publisher

Overview.....	A-1
---------------	-----

## B XML Publisher Configuration File

XML Publisher Configuration File.....	B-1
Structure.....	B-3
Properties.....	B-3
List of Available Properties.....	B-4
Font Definitions.....	B-6

Font Fallback Mechanism..... B-8

**C Moving Templates and Data Definitions Between E-Business Suite Instances**

Overview..... C-1

**D Oracle Report to XML Publisher Report Migration**

Overview..... D-1

**Index**

---

# Send Us Your Comments

**Oracle XML Publisher Administration and Developer's Guide, Release 11i**

**Part No. E05321-01**

Oracle welcomes customers' comments and suggestions on the quality and usefulness of this document. Your feedback is important, and helps us to best meet your needs as a user of our products. For example:

- Are the implementation steps correct and complete?
- Did you understand the context of the procedures?
- Did you find any errors in the information?
- Does the structure of the information help you with your tasks?
- Do you need different information or graphics? If so, where, and in what format?
- Are the examples correct? Do you need more examples?

If you find any errors or have any other suggestions for improvement, then please tell us your name, the name of the company who has licensed our products, the title and part number of the documentation and the chapter, section, and page number (if available).

Note: Before sending us your comments, you might like to check that you have the latest version of the document and if any concerns are already addressed. To do this, access the new Applications Release Online Documentation CD available on Oracle MetaLink and [www.oracle.com](http://www.oracle.com). It contains the most current Documentation Library plus all documents revised or released recently.

Send your comments to us using the electronic mail address: [appsdoc\\_us@oracle.com](mailto:appsdoc_us@oracle.com)

Please give your name, address, electronic mail address, and telephone number (optional).

If you need assistance with Oracle software, then please contact your support representative or Oracle Support Services.

If you require training or instruction in using Oracle software, then please contact your Oracle local office and inquire about our Oracle University offerings. A list of Oracle offices is available on our Web site at [www.oracle.com](http://www.oracle.com).



---

# Preface

## Intended Audience

Welcome to Release 11*i* of the *Oracle XML Publisher Administration and Developer's Guide*.

This guide is intended for administrators and developers. The administration topics assume you have a working knowledge of Oracle Applications System Administration and an understanding of your system's specific implementation. The developer's topics assume you have an understanding of Java programming, XSL, and XML technologies.

See Related Information Sources on page x 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 Introduction
- 2 Using the Template Manager
- 3 Generating Your Customized Report
- 4 Administration
- 5 Data Templates
- 6 Calling XML Publisher APIs
- 7 Using the Delivery Manager APIs
- 8 Integrating the Document Viewer into an Application
- A Setting Up XML Publisher
- B XML Publisher Configuration File
- C Moving Templates and Data Definitions Between E-Business Suite Instances
- D Oracle Report to XML Publisher Report Migration

## Related Information Sources

This book is included on the Oracle Applications Documentation Library.. You can download soft-copy documentation as PDF files from the Oracle Technology Network at <http://otn.oracle.com/documentation>, or you can purchase hard-copy documentation from the Oracle Store at <http://oraclestore.oracle.com>. The Oracle Applications Documentation Library contains the latest information. If substantial changes to this book are necessary, a revised version will be made available on the "virtual" documentation library on Oracle*MetaLink*.

If this guide refers you to other Oracle Applications documentation, use only the latest versions of those guides.

### Online Documentation

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

- **Online Help** - Online help patches (HTML) are available on Oracle*MetaLink*.
- **PDF Documentation** - See the Oracle Applications Documentation Library for

current PDF documentation for your product with each release. The Oracle Applications Documentation Library is also available on *OracleMetaLink* and is updated frequently.

- **Oracle Electronic Technical Reference Manual** - The Oracle Electronic Technical Reference Manual (eTRM) contains database diagrams and a detailed description of database tables, forms, reports, and programs for each Oracle Applications product. This information helps you convert data from your existing applications and integrate Oracle Applications data with non-Oracle applications, and write custom reports for Oracle Applications products. The Oracle eTRM is available on *Oracle MetaLink*.

## **Oracle Applications User's Guide**

This guide explains how to navigate, enter data, query, and run reports using the user interface (UI) of Oracle Applications. This guide also includes information on setting user profiles, as well as running and reviewing concurrent requests

## **Oracle Applications Developer's Guide**

This guide contains the coding standards followed by the Oracle Applications development staff. It describes the Oracle Application Object Library components needed to implement the Oracle Applications user interface described in the *Oracle Applications User Interface Standards for Forms-Based Products*. It also provides information to help you build your custom Oracle Forms Developer forms so that they integrate with Oracle Applications.

## **Oracle Applications Flexfields Guide**

This guide provides flexfields planning, setup, and reference information for the Oracle Applications implementation team, as well as for users responsible for the ongoing maintenance of Oracle Applications product data. This guide also provides information on creating custom reports on flexfields data.

## **Oracle Application Framework Developer's Guide**

This guide contains the coding standards followed by the Oracle Applications development staff to produce applications built with Oracle Application Framework. This guide is available in PDF format on *OracleMetaLink* and as online documentation in JDeveloper 10g with Oracle Application Extension.

## **Oracle Applications Concepts**

This book is intended for all those planning to deploy Oracle E-Business Suit, or contemplating significant changes to a configuration. After describing the Oracle

Applications architecture and technology stack, it focuses on strategic topics, giving a broad outline of the actions needed to achieve a particular goal, plus the installation and configuration choices that may be available.

## **Oracle Applications System Administrator's Guide Documentation Set**

This documentation set provides planning and reference information for the Oracle Applications System Administrator. *Oracle Applications System Administrator's Guide - Configuration* contains information on system configuration steps, including defining concurrent programs and managers, enabling Oracle Applications Manager features, and setting up printers and online help. *Oracle Applications System Administrator's Guide - Maintenance* provides information for frequent tasks such as monitoring your system with Oracle Applications Manager, managing concurrent managers and reports, using diagnostic utilities, managing profile options, and using alerts. *Oracle Applications System Administrator's Guide - Security* describes User Management, data security, function security, auditing, and security configurations.

## **Oracle Applications Multiple Organizations Implementation Guide**

This guide describes the multiple organizations concepts in Oracle Applications. It describes in detail on setting up and working effectively with multiple organizations in Oracle Applications.

## **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.

---

# Introduction

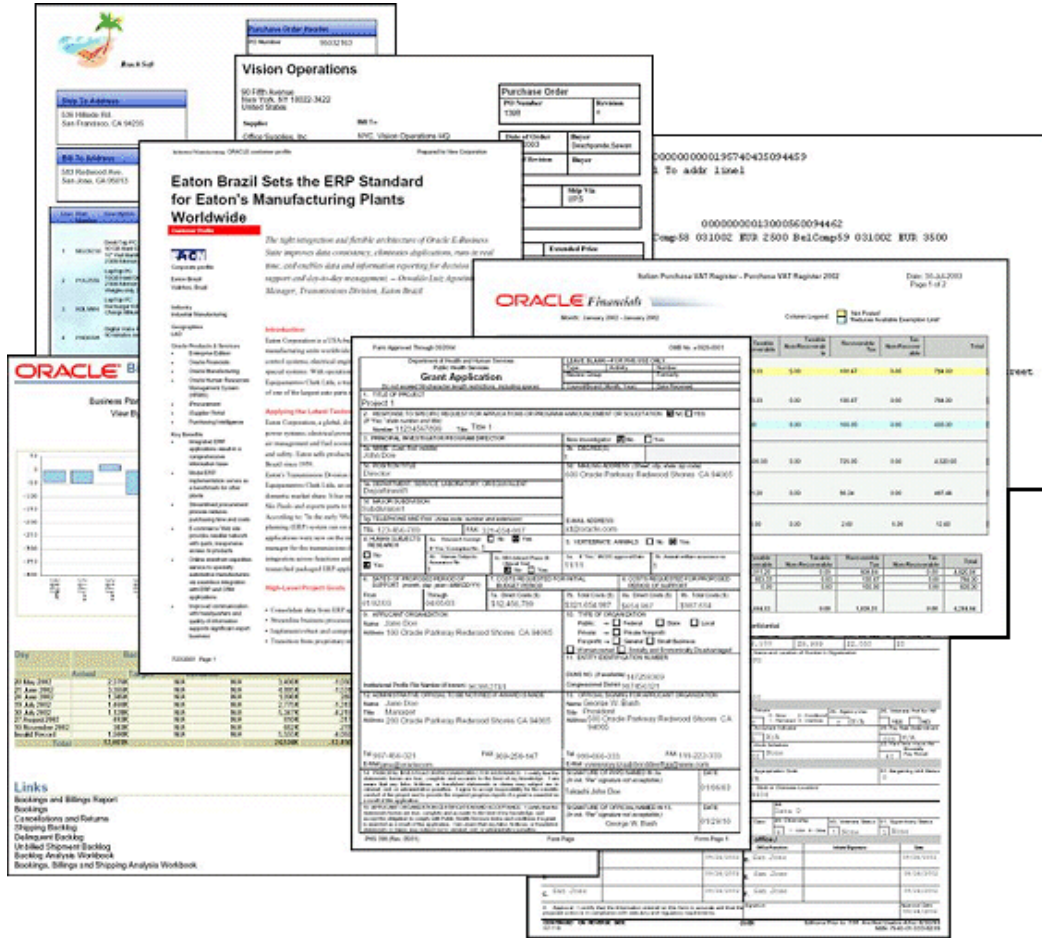
## What is XML Publisher?

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:



## Structure of the XML Publisher Documentation Set

The XML Publisher documentation set contains the following two guides: *Oracle XML Publisher Report Designer's Guide* and the *Oracle XML Publisher Administration and Developer's Guide*.

### Oracle XML Publisher Administration and Developer's Guide

This guide includes information on setting up XML Publisher, running reports, using the data engine, and leveraging the APIs.

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.

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.

Oracle Report to XML Publisher Report Migration - describes how to use the conversion utility to convert existing Oracle Reports to XML Publisher reports.

The *Oracle XML Publisher Report Designer's Guide* provides instructions for designing report layout templates. It includes the following chapters:

### **Oracle XML Publisher Report Designer's Guide**

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.

Creating an eText Template - 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.

XML Publisher Extended Functions - lists SQL and XSL functions that XML Publisher has extended.

Supported XSL-FO Elements - lists the FO elements supported by the XML Publisher engines.



---

## Using the Template Manager

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.

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 register a template in the Template Manager:

1. Create a Data Definition for your template, page 2-2 in the Template Manager.
2. Register the layout template file, page 2-5.

### 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 (in the next section), you must assign it a data definition that already 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.

The screenshot shows the Oracle XML Publisher interface. At the top, there is a navigation bar with tabs for 'Templates', 'Data Definitions', and 'Administration'. The 'Data Definitions' tab is active. Below the navigation bar, there is a section titled 'Create Data Definition'. The form contains the following fields:

- \* Name: Simple Customer Listing
- \* Application: Receivables
- End Date: (empty)
- \* Code: RAXCUS
- \* Start Date: 03-15-2007

There are 'Cancel' and 'Apply' buttons at the top right of the form. Below the form fields is a large text area labeled 'Description'.

- |                    |                                                                                                                                                                                                                                                                                                                                                                                      |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>        | Enter a user-friendly name for your data definition.                                                                                                                                                                                                                                                                                                                                 |
| <b>Code</b>        | If you are using the Oracle Applications concurrent manager to generate your report, the data definition <b>Code</b> 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. |
| <b>Application</b> | Select the report's application from the LOV.                                                                                                                                                                                                                                                                                                                                        |
| <b>Start Date</b>  | Enter the date from which the data definition will be active.                                                                                                                                                                                                                                                                                                                        |
| <b>End Date</b>    | You cannot delete data definitions from the Template                                                                                                                                                                                                                                                                                                                                 |

Manager. To make the data definition inactive, enter an end date.

Select **Apply** to create the data definition. A confirmation message will display to indicate that you have successfully created your data definition.

The screenshot shows the Oracle XML Publisher interface. At the top, there is a navigation bar with 'ORACLE XML Publisher' on the left and 'Home', 'Logout', 'Preferences', 'Help', and 'Diagnostics' on the right. Below this is a secondary navigation bar with 'Templates', 'Data Definitions', and 'Administration'. The main content area is titled 'Data Definitions' and features a green confirmation banner that reads: 'Data Definition Simple Customer Listing has been successfully created.' Below the banner is a link: 'View Data Definition: Simple Customer Listing'. The 'General' section contains a table with the following information:

Name	Simple Customer Listing	Code	RAXCUS	<input type="button" value="Update"/>	<input type="button" value="Edit Configuration"/>
Application	Receivables	Start Date	03-15-2007		
End Date					

Below the table is a 'Description' field with a large empty text area. At the bottom, the 'Files' section has three buttons: 'XML Schema Add File', 'Data Template Add File', and 'Preview Data Add File'.

You can now optionally add the following to complete your Data Definition:

#### 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.

**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*.

<b>Data Template</b>	If you are using an XML Publisher data template to generate the data for this data definition, use the <b>Add File</b> button to upload your data template. For information on creating data templates, see <i>Data Templates</i> , page 5-1 .
<b>Preview Data</b>	To use the report <b>Preview</b> feature of the Template Manager, upload a sample XML file from the data source. The <b>Preview</b> feature is available from the <b>View Template page</b> , page 2-8 and also at runtime from the Oracle Applications request submission interface.
<b>Edit Configuration</b>	Select this button to add configuration instructions specific to this Data Definition. Configurations defined at this level will take precedence over site-level configurations, but will be overridden by template-level configurations. For more information, see <i>Setting Configuration Properties</i> , page 4-2.

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 2-2.

## 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 4-2.

## 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 2-7). To upload additional template files or to change the Default Template File, use the **View Template** page (see *Viewing and Updating a Template*, page 2-8).

If your template type is PDF, the **Template Mapping** region will display after you click the **Apply** button. See *Template Mapping*, page 2-7.

Templates

Create Template

Start Date and End Date define when the Template is active. The Name can be translated while the Code is identical for all languages.

Cancel Apply

General

* Name	<input type="text" value="Customer Listing"/>	* Code	<input type="text" value="AR_CustomerListing"/>
* Application	<input type="text" value="Receivables"/>	* Data Definition	<input type="text" value="Simple Customer Listing"/>
* Type	<input type="text" value="RTF"/>	* Start Date	<input type="text" value="03-15-2007"/>
		End Date	<input type="text"/>
		Subtemplate	<input type="text" value="No"/>

Description

Template File

* File	<input type="text" value="CustomerListing.rtf"/>	<input type="button" value="Browse..."/>
* Language	<input type="text" value="English"/>	
Territory	<input type="text" value="United States"/>	

Translatable

Cancel Apply

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 2-8.

- 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 2-2.
- 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. For example, you may create a subtemplate to contain common content that you want shared across reports so that you do not have to duplicate that content in all the templates. You enter syntax in the primary template to "call" the subtemplate so that at runtime its contents are included in the report. For more information see *Using Subtemplates, Oracle XML Publisher Report Designer's Guide*.

<b>File</b>	Use the <b>Browse</b> button to upload your template layout file.
<b>Language</b>	Select the template language.  Add more language template files to your template definition from the <b>View Template</b> page. See <i>Adding Templates for Additional Languages</i> , page 2-10.
<b>Territory</b>	Select the language territory.
<b>Translatable (check box)</b>	Select this check box if you want this template to be translatable. Only RTF templates are translatable. For more information see <i>Translatable Templates</i> , page 2-12.

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 2-11.

For information on creating placeholders in the PDF template, see Creating a Placeholder, *Oracle XML Publisher Report Designer's Guide*.

## 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.

<b>Code</b>	Assign a template <b>Code</b> using the product short name and a descriptive ending.
<b>Name</b>	Enter a user-friendly name for your template.
<b>Application</b>	Select the report's application from the LOV.
<b>Source Template Name</b>	(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

### General

Name **Customer Listing** Code **AR\_CustomerListing** [Update](#) [Edit Configuration](#)  
 Application **Receivables** Data Definition **Simple Customer Listing**  
 Type **RTF** Start Date **17-Dec-2004**  
 Default File **CustomerListing.rtf** End Date  
 Default File Language **English** Subtemplate No  
 Default File Territory **United States**

Description

### Template Files

Preview Format  [Add File](#)

#### Localized Templates

File Name	Language	Territory	Preview	Download	Update	Delete
CustomerListingJapan.rtf	Japanese	Japan				

#### Translatable Template

[Export Translation](#) [Upload Translations](#)

File Name	Language	Territory	Preview	Download	Update	Delete
CustomerListing.rtf	English	United States				

#### Available Translations

[Select Translation\(s\) and...](#) [Enable](#) [Disable](#)  
[Select All](#) | [Select None](#)

Select	Status	Language	Territory	Preview	Export Translation	Progress
<input type="checkbox"/>	✓	French	France			✓ Complete

From the **View Template** page, you can:

- Update the general definitions, page 2-10
- Preview the template, page 2-10
- Download the template file, page 2-10
- Update the template file for editing, page 2-10
- Add localized template files for additional languages, page 2-10
- Export the XLIFF file for translation of translatable templates (RTF templates only), page 2-12
- Upload the translated XLIFF files (RTF templates only), page 2-16
- Enable or Disable available translations (RTF templates only), page 2-17
- Update the template field mapping (PDF templates only), page 2-11
- Set runtime properties for a template, page 2-12

**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 2-8 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 2-8.

## 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, Oracle XML Publisher Report Designer's Guide* or *Creating a PDF Template, Oracle XML Publisher Report Designer's Guide*.

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 2-12.

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

If mapping is required, map each Template Field to a Data Source Element.

Template Field Name	Data Source Element
ITEM_DESCRIPTION	CLASSIFIED_FLAG
LINE_NUMBER	L_SMALL_BUSINESS_FLAG
LINE_PROJECT_NAME	LINE_COST_OF_MONEY
LINE_QUANTITY	L_CONTACT_MAJOR_VERSION
LINE_START_DATE	SCHEDULED_DELV_DEFAULT
LINE_STATUS	PARTY_NAME
LINE_STYLE	L_CONTACT_ATTRIBUTE10
SUBCONTRACTED_FLAG	L_CONTACT_ATTRIBUTE11
TEXT1	L_CONTACT_ATTRIBUTE12
TEXT2	DESCRIPTION
	L_CONTACT_ATTRIBUTE13
	MINORITY_GROUP_LOOKUP_CODE

Previous Next 10

Cancel Apply

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 4-2.

## 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 2-10.

### 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 2-15.

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:

```
<xliff>
  <file>
    <header>
      <body>
        <trans-unit>
          <source>
          <target>
          <note>
```

The following figure shows an excerpt from an untranslated XLIFF file:

```

source-language Attribute      target-language Attribute      Embedded Data Field
<?xml version = '1.0' encoding = 'utf-8' ?>
<xliff version="1.0">
  <file source-language="en-US" target-language="en-US" datatype="XDO" original="orpher"
    <header/>
    <body>
      <trans-unit id="d678c24b" maxbytes="4000" maxwidth="90" size-unit="char" transl
source and target elements
      <source>Italian Purchase VAT Register - [&#1]</source>
      <target>Italian Purchase VAT Register - [&#1]</target>
      <note>Text located: header/table, token &#1:anonymous placeholder(s)</not
      </trans-unit>
      <trans-unit id="4d3eb24" maxbytes="4000" maxwidth="15" size-unit="char" transl
      <source>Total</source>
      <target>Total</target>
      <note>Text located: body/table</note>
      </trans-unit>
      <trans-unit id="aeccc17e" maxbytes="4000" maxwidth="37" size-unit="char" transl
      <source>Non-Recoverable</source>
      <target>Non-Recoverable</target>
      <note>Text located: body/table</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,

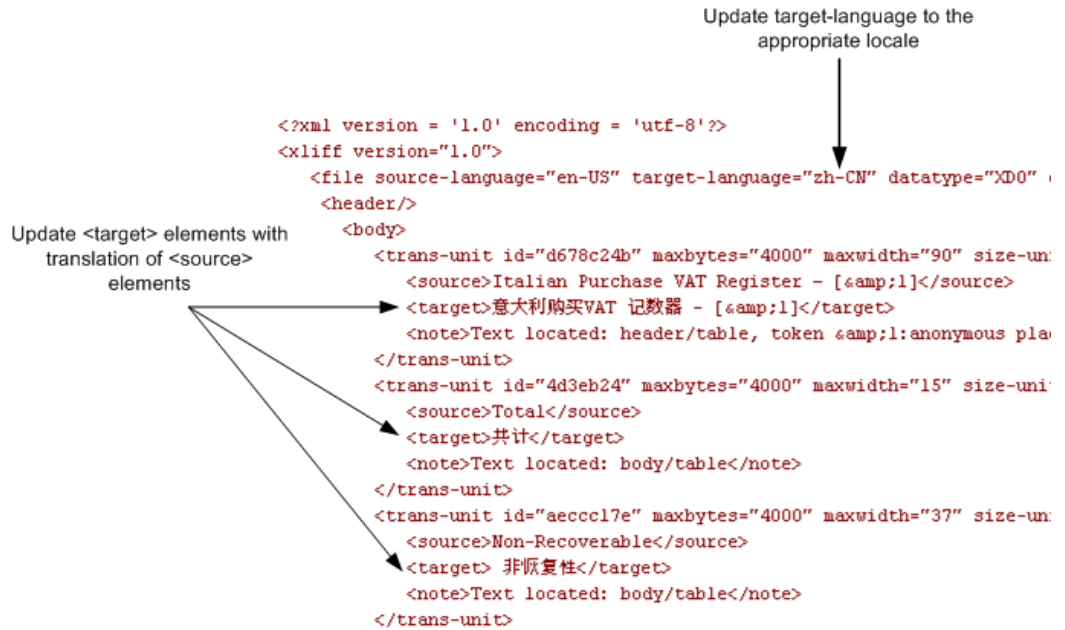
```
<source>Total</source>
```

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>
```

The following figure shows the sample XLIFF file from the previous figure updated with the Chinese translation:



### 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:

[ & 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

```
<source>Italian Purchase VAT Register - [&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

**Note:** For more information on the International Organization for Standardization (ISO) and the code lists, see International Organization for Standardization [http://www.iso.org/iso/en/ISOOnline.frontpage].

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:

Translation (Language/Territory)	source-language value	target-language value
From English/US To English/Canada	en-US	en-CA
From English/US To Chinese/China	en-US	zh-CN
From Japanese/Japan To French/France	ja-JP	fr-FR

## 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

Select	Status	Language	Territory	Preview	Export Translation	Progress
<input type="checkbox"/>	✓	Chinese	China			✓ Complete

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 covers the following topics:

- Using the Concurrent Manager to Generate Your Custom Output

## Using the Concurrent Manager to Generate Your Custom Output

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**.

## Publishing Process

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.

---

# Administration

This chapter covers the following topics:

- XML Publisher Administration
- Setting Runtime Properties
- 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 4-2

**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 4-3.

- Define font mappings, page 4-13
- Upload font files, page 4-13

- Define currency formats, page 4-21

## Setting Runtime Properties

ORACLE XML Publisher

Home Logout Preferences Personalize Page Diagnostics

Templates Data Definitions Administration

Configuration Font Mappings Font Files Currencies

Configuration

Reset Save

Expand All | Collapse All

Focus	Properties	Site Value
	▼ Properties	
⊕	▶ General	
⊕	▶ PDF Output	
⊕	▶ PDF Security	
⊕	▶ RTF Output	
⊕	▶ HTML Output	
⊕	▶ FO Processing	
⊕	▶ RTF Templates	
⊕	▶ PDF Templates	
⊕	▶ XLIFF Extraction	

Reset Save

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 2-12. For information on setting properties at the Data Definition level, see *Setting Runtime Properties for a Data Definition*, page 2-5.

### 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 B-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	<p>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.</p> <p><b>Note:</b> 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 B-1 for details on setting up this file.</p>

## PDF Output Properties

The following properties are available for PDF output:

Property Name	Internal Name	Default Value	Description
Compress PDF output	pdf-compressi on	True	Specify "True" or "False" to control compression of the output PDF file.
Hide PDF viewer's menu bars	pdf-hide-menu bar	False	Specify "True" to hide the viewer application's menu bar when the document is active. The menu bar option is only effective when using the Export button, which displays the output in a standalone Acrobat Reader application outside of the browser.
Replace smart quotes	pdf-replace-s martquotes	True	Set to "False" if you do not want curly quotes replaced with straight quotes in your PDF output.

## PDF Security

Use the following properties to control the security settings for your output PDF

documents:

Property Name	Internal Name	Default Value	Description
Enable PDF Security	pdf-security	False	If you specify "True," the output PDF file will be encrypted. You must also specify the following properties: <ul style="list-style-type: none"><li>• Open document password</li><li>• Modify permissions password</li><li>• Encryption Level</li></ul>
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".

Property Name	Internal Name	Default Value	Description
Encryption level	pdf-encryption-level	0 - low	<p>Specify the encryption level for the output PDF file. The possible values are:</p> <ul style="list-style-type: none"> <li>• 0: Low (40-bit RC4, Acrobat 3.0 or later)</li> <li>• 1: High (128-bit RC4, Acrobat 5.0 or later)</li> </ul> <p>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:</p> <ul style="list-style-type: none"> <li>• Disable printing</li> <li>• Disable document modification</li> <li>• Disable context copying, extraction, and accessibility</li> <li>• Disable adding or changing comments and form fields</li> </ul> <p>When Encryption level is set to 1, the following properties are available:</p> <ul style="list-style-type: none"> <li>• Enable text access for screen readers</li> <li>• Enable copying of text, images, and other content</li> <li>• Allowed change level</li> <li>• Allowed printing level</li> </ul>
Disable printing	pdf-no-printing	False	<p>Permission available when "Encryption level" is set to 0. When set to "True", printing is disabled for the PDF file.</p>
Disable document modification	pdf-no-changing-the-document	False	<p>Permission available when "Encryption level" is set to 0. When set to "True", the PDF file cannot be edited.</p>

Property Name	Internal Name	Default Value	Description
Disable context copying, extraction, and accessibility	pdf-no-cceda	False	Permission available when "Encryption level" is set to 0. When set to "True", the context copying, extraction, and accessibility features are disabled.
Disable adding or changing comments and form fields	pdf-no-accff	False	Permission available when "Encryption level" is set to 0. When set to "True", the ability to add or change comments and form fields is disabled.
Enable text access for screen readers	pdf-enable-accessibility	True	Permission available when "Encryption level" is set to 1. When set to "True", text access for screen reader devices is enabled.
Enable copying of text, images, and other content	pdf-enable-copying	False	Permission available when "Encryption level" is set to 1. When set to "True", copying of text, images, and other content is enabled.
Allowed change level	pdf-changes-allowed	0	<p>Permission available when "Encryption level" is set to 1. Valid Values are:</p> <ul style="list-style-type: none"> <li>• 0: none</li> <li>• 1: Allows inserting, deleting, and rotating pages</li> <li>• 2: Allows filling in form fields and signing</li> <li>• 3: Allows commenting, filling in form fields, and signing</li> <li>• 4: Allows all changes except extracting pages</li> </ul>

Property Name	Internal Name	Default Value	Description
Allowed printing level	pdf-printing-allowed	0	<p>Permission available when "Encryption level" is set to 1. Valid values are:</p> <ul style="list-style-type: none"> <li>• 0: None</li> <li>• 1: Low resolution (150 dpi)</li> <li>• 2: High resolution</li> </ul>

## RTF Output

The following properties can be set to govern RTF output files:

Property Name	Internal Name	Default Value	Description
Enable change tracking	rtf-track-changes	False	Set to "True" to enable change tracking in the output RTF document.
Protect document for tracked changes	rtf-protect-document-for-tracked-changes	False	Set to "True" to protect the document for tracked changes.

## HTML Output

The following properties can be set to govern HTML output files:

Property Name	Internal Name	Default Value	Description
Base image URI	html-image-base-uri	N/A	Base URI which is inserted into the <code>src</code> attribute of the image tag before the image file name. This works only when the image is embedded in the template.
Image file directory	html-image-dir	N/A	Enter the directory for XML Publisher to store the image files that are embedded in the template.

Property Name	Internal Name	Default Value	Description
Base CSS URI	html-css-base-uri	N/A	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 <code>make-accessible</code> is true.
CSS file directory	html-css-dir	N/A	The CSS directory where XML Publisher stores the css file. You must set this property when <code>make-accessible</code> is true.
Show header	html-show-header	True	Set to "False" to suppress the template header in HTML output.
Show footer	html-show-footer	True	Set to "False" to suppress the template footer in HTML output.
Replace smart quotes	html-replace-smartquotes	True	Set to "False" if you do not want curly quotes replaced with straight quotes in your HTML output.
Character set	html-output-character-set	UTF-8	Specify the output HTML character set.
Make HTML output accessible	make-accessible	False	Specify true if you want to make the HTML output accessible.

## FO Processing Properties

The following properties can be set to govern FO processing:

Property Name	Internal Name	Default Value	Description
Font mapping set	N/A	N/A	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 <i>Creating a Font Mapping</i> , page 4-14 for more information.

Property Name	Internal Name	Default Value	Description
Currency format Set	N/A	N/A	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 <i>Defining Currency Formats</i> , for more information.
Bidi language digit substitution type	digit-substitution	None	Valid values are "None" and "National". When set to "None", Eastern European numbers will be used. When set to "National", Hindi format (Arabic-Indic digits) will be used. This setting is effective only when the locale is Arabic, otherwise it is ignored.
Pages cached during processing	system-cache-page-size	50	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.
Disable variable header support	fo-prevent-variable-header	False	If "True", prevents variable header support. Variable header support automatically extends the size of the header to accommodate the contents.
Add prefix to IDs when merging FO	fo-merge-conflict-resolution	False	When merging multiple XSL-FO inputs, the FO Processor automatically adds random prefixes to resolve conflicting IDs. Setting this property to "True" disables this feature.
Use XML Publisher's XSLT processor	xslt-xdoparser	True	Controls XML Publisher's parser usage. If set to False, XSLT will not be parsed.
Enable scalable feature of XSLT processor	xslt-scalable	False	Controls the scalable feature of the XDO parser. The property "Use XML Publisher's XSLT processor" must be set to "True" for this property to be effective.

Property Name	Internal Name	Default Value	Description
Enable XSLT runtime optimization	xslt-runtime-optimization	True	<p>When set to "True", the overall performance of the FO processor is increased and the size of the temporary FO files generated in the temp directory is significantly decreased. Note that for small reports (for example 1-2 pages) the increase in performance is not as marked.</p> <p>To further enhance performance when you set this property to True, it is recommended that you set the property <b>Extract attribute sets</b> to "False". See RTF Template Properties, page 4-10.</p>

## RTF Template Properties

The following properties can be set to govern RTF templates:

Property Name	Internal Name	Default Value	Description
Extract attribute sets	rtf-extract-attribute-sets	Auto	<p>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.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> <li>• Enable - extract attribute sets for all templates and subtemplates</li> <li>• Auto - extract attribute sets for templates, but not subtemplates</li> <li>• Disable - do not extract attribute sets</li> </ul>

Property Name	Internal Name	Default Value	Description
Enable XPath rewriting	rtf-rewrite-path	True	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	rtf-checkbox-glyph	Default value: Albany WT J;9746;9747/A	<p>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: <code>fontname;&lt;unicode font number for true value's glyph &gt;;&lt;unicode font number for false value's glyph&gt;</code></p> <p>Example: Albany WT J;9746;9747/A</p> <p>Note that the font that you specify must be made available to XML Publisher at runtime.</p>

## PDF Template Properties

The following properties can be set to govern PDF templates:

Property Name	Default Value	Description
Font mapping set	N/A	Select the Font Mapping Set. This will be used for mapping fonts from PDF templates to output PDF documents. See <i>Creating a Font Mapping</i> , page 4-14 for more information.

## XLIFF Extraction

The following properties can be set to govern XLIFF extraction:

Property Name	Internal Name	Default Value	Description
Translation expansion percentage	<code>xliff-trans-expansion</code>	150 (percentage)	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.
Minimum translation length	<code>xliff-trans-min-length</code>	15 (characters)	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.
Maximum translation length	<code>xliff-trans-max-length</code>	4000 (characters)	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.
Extract white space	<code>xliff-trans-null</code>	False	Instructs the XLIFF extractor to create a translation unit for a record that contains only spaces (is null). Set to "True" to generate the translation unit.
Extract sections without letters	<code>xliff-trans-symbol</code>	False	Instructs the XLIFF extractor whether to extract symbol characters. If set to "False" only A-Z and a-z will be extracted.
Extract words with underscores	<code>xliff-trans-keyword</code>	True	If set to "False", words with underscores will not be extracted.

## Uploading Font Files

The screenshot shows the Oracle XML Publisher Administration interface. At the top, there is the Oracle XML Publisher logo and navigation icons for Home, Logout, Preferences, Personalize Page, and Diagnostics. Below this is a navigation bar with tabs for Templates, Data Definitions, and Administration. The current page is 'Font Files', which is highlighted in the navigation bar. Under 'Font Files', there is a 'Search' section with input fields for 'Font Name' and 'File Name', and a 'Go' button. To the right of the search section is a 'Create Font File' button. Below the search section is a table with three columns: 'Font Name', 'File Name', and 'Update'. The table contains five rows of font data. At the bottom of the page, there is a footer with navigation links for Templates, Data Definitions, Administration, Home, Logout, Preferences, Personalize Page, and Diagnostics, along with a copyright notice for Oracle Corporation.

Font Name	File Name	Update
Batang TTC	batang.ttc	
Comic Sans MS	CL.log	
JY Collection	batang.ttc	
JY Test Font	impact.ttf	
Test font	5aaccdf.ttf	

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 4-8 for setting the FO to PDF mapping. See PDF Template Properties, page 4-11 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 4-15 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 4-17.
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 4-18 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 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:

Level	Font Family	Style	Weight	Language	Territory	Target Font
Site	Times New Roman	normal	normal	(none)	(none)	Times
Site	Arial	normal	normal	Japanese	Japan	Times
Template	Arial	normal	normal	Japanese	(none)	Courier

Level	Font Family	Style	Weight	Language	Territory	Target Font
Template	Trebuchet MS	normal	normal	(none)	(none)	Helvetica

At runtime if the locale of the template file is Japanese/Japan, the following font mappings will be used:

Font Family	Style	Weight	Target Font
Times New Roman	normal	normal	Times
Arial	normal	normal	Times
Trebuchet MS	normal	normal	Helvetica

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 two task}/`

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 4-3 for information on setting the temporary directory.

## Predefined Fonts

XML Publisher provides a set of Type1 fonts and a set of TrueType fonts. You can select any of these fonts as a target font with no additional setup required.

The Type1 fonts are listed in the following table:

### Type 1 Fonts

Number	Font Family	Style	Weight	Font Name
1	serif	normal	normal	Time-Roman
1	serif	normal	bold	Times-Bold
1	serif	italic	normal	Times-Italic
1	serif	italic	bold	Times-BoldItalic
2	sans-serif	normal	normal	Helvetica
2	sans-serif	normal	bold	Helvetica-Bold
2	sans-serif	italic	normal	Helvetica-Oblique
2	sans-serif	italic	bold	Helvetica-BoldOblique
3	monospace	normal	normal	Courier
3	monospace	normal	bold	Courier-Bold
3	monospace	italic	normal	Courier-Oblique
3	monospace	italic	bold	Courier-BoldOblique
4	Courier	normal	normal	Courier
4	Courier	normal	bold	Courier-Bold
4	Courier	italic	normal	Courier-Oblique
4	Courier	italic	bold	Courier-BoldOblique
5	Helvetica	normal	normal	Helvetica
5	Helvetica	normal	bold	Helvetica-Bold
5	Helvetica	italic	normal	Helvetica-Oblique

Number	Font Family	Style	Weight	Font Name
5	Helvetica	italic	bold	Helvetica-BoldOblique
6	Times	normal	normal	Times
6	Times	normal	bold	Times-Bold
6	Times	italic	normal	Times-Italic
6	Times	italic	bold	Times-BoldItalic
7	Symbol	normal	normal	Symbol
8	ZapfDingbats	normal	normal	ZapfDingbats

The TrueType fonts are listed in the following table. All TrueType fonts will be subsetted and embedded into PDF.

Number	Font Family Name	Style	Weight	Actual Font	Actual Font Type
1	Albany WT	normal	normal	ALBANYWT.ttf	TrueType (Latin1 only)
2	Albany WT J	normal	normal	ALBANWTJ.ttf	TrueType (Japanese flavor)
3	Albany WT K	normal	normal	ALBANWTK.ttf	TrueType (Korean flavor)
4	Albany WT SC	normal	normal	ALBANWTS.ttf	TrueType (Simplified Chinese flavor)
5	Albany WT TC	normal	normal	ALBANWTT.ttf	TrueType (Traditional Chinese flavor)

<b>Number</b>	<b>Font Family Name</b>	<b>Style</b>	<b>Weight</b>	<b>Actual Font</b>	<b>Actual Font Type</b>
6	Andale Duospace WT	normal	normal	ADUO.ttf	TrueType (Latin1 only, Fixed width)
6	Andale Duospace WT	bold	bold	ADUOB.ttf	TrueType (Latin1 only, Fixed width)
7	Andale Duospace WT J	normal	normal	ADUOJ.ttf	TrueType (Japanese flavor, Fixed width)
7	Andale Duospace WT J	bold	bold	ADUOJB.ttf	TrueType (Japanese flavor, Fixed width)
8	Andale Duospace WT K	normal	normal	ADUOK.ttf	TrueType (Korean flavor, Fixed width)
8	Andale Duospace WT K	bold	bold	ADUOKB.ttf	TrueType (Korean flavor, Fixed width)
9	Andale Duospace WT SC	normal	normal	ADUOSC.ttf	TrueType (Simplified Chinese flavor, Fixed width)
9	Andale Duospace WT SC	bold	bold	ADUOSCB.ttf	TrueType (Simplified Chinese flavor, Fixed width)
10	Andale Duospace WT TC	normal	normal	ADUOTC.ttf	TrueType (Traditional Chinese flavor, Fixed width)

Number	Font Family Name	Style	Weight	Actual Font	Actual Font Type
10	Andale Duospace WT TC	bold	bold	ADUOTCB.ttf	TrueType (Traditional Chinese flavor, Fixed width)

## Defining Currency Formats

The screenshot shows the Oracle XML Publisher Administration interface. The top navigation bar includes links for Home, Logout, Preferences, Help, and Contact Admin. The main navigation bar has tabs for Templates, Data Definitions, and Administration. The Administration tab is active, and the sub-navigation bar shows Configuration, Font Mappings, Font Files, and Currencies. The Currencies page is displayed, showing the general configuration for the 'European Set' with the code 'Euro\_set'. Below this is a table of currency formats with columns for Currency Name, Currency Code, Format Mask, Update, and Delete. The table lists Pound Sterling (GBP, 999D99), Euro (EUR, 999D999), Swiss Franc (CHF, 9G999D99), and Swedish Krona (SEK, 9G999D99).

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 4-8 list.
4. Enter the `format-currency` command in your RTF template to apply the format to the field at runtime. See *Currency Formatting, Oracle XML Publisher Report*

*Designer's Guide.*

**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, *Oracle XML Publisher Report Designer's Guide* 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.

---

## Data Templates

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 any 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
- BLOB and CLOB datatype selection
- 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.

## Overview of Implementing a Data Template for Use with the Concurrent Manager

The process for implementing a data template to be called by the Concurrent Manager is as follows (this chapter covers each step in more detail):

### Use an XML or text editor to:

- Write the data template XML document following the guidelines in this chapter.

### Use the Template Manager to:

- 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.

### Use Oracle Applications System Administrator responsibility to:

- 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

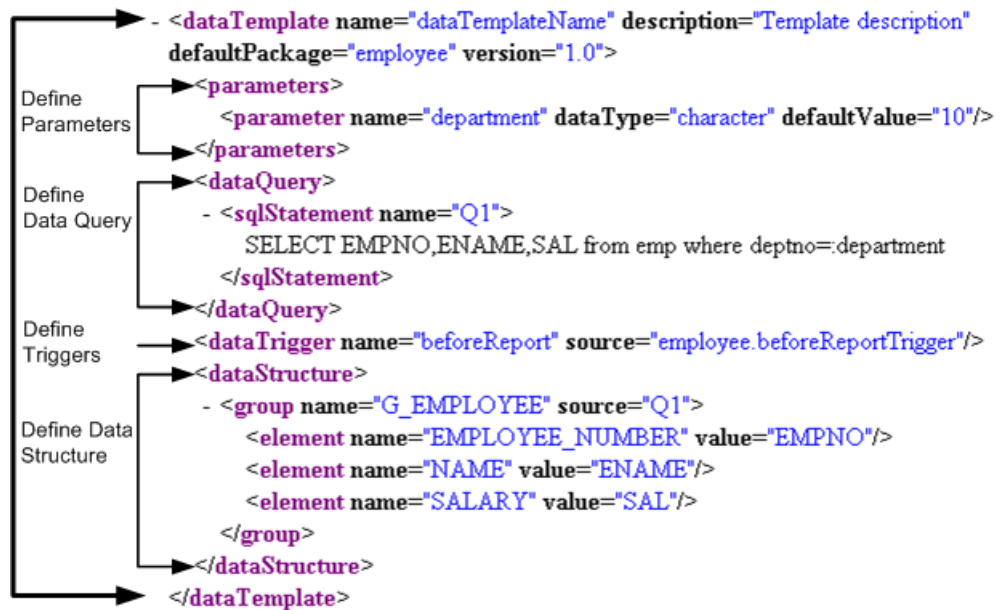
corresponding data template in the Template Manager.

- 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.

Element	Attributes/Description
dataTemplate (Required)	Attributes: <ul style="list-style-type: none"> <li>• name (Required)</li> <li>• description</li> <li>• version (Required)</li> <li>• defaultPackage - the PL/SQL package name to resolve any lexical references, group filters, or data triggers defined in the template.</li> <li>• dataSourceRef - the default data source reference for the entire data template. It is required only when performing a distributed query across multiple data sources.</li> </ul>
properties	Consists of one or more <property> elements to support the XML output and Data Engine specific properties.
property	Attributes: <ul style="list-style-type: none"> <li>• name (Required) - the property name.</li> <li>• value - valid values for this property.</li> </ul>
parameters	Consists of one or more <parameter> elements.
parameter	Attributes: <ul style="list-style-type: none"> <li>• name (Required) - the parameter name that will be referenced in the template.</li> <li>• dataType - valid values are: "character", "date", "number"</li> <li>• defaultValue - value to use for the parameter if none supplied from the data</li> <li>• include_in_output - whether this parameter should appear in the XML output or not. The valid values are "true" and "false".</li> </ul>
lexicals	Consists of one or more lexical elements to support flexfields.

Element	Attributes/Description
lexical	<p>There are four types of key flexfield-related lexicals as follows:</p> <ul style="list-style-type: none"> <li>• oracle.apps.fnd.flex.kff.segments_metadata</li> <li>• oracle.apps.fnd.flex.kff.select</li> <li>• oracle.apps.fnd.flex.kff.where</li> <li>• oracle.apps.fnd.flex.kff.order_by</li> </ul>
dataQuery (Required)	Consists of one or more <sqlstatement> or <xml>elements.
sqlstatement (Required)	<p>Attributes:</p> <ul style="list-style-type: none"> <li>• 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.</li> <li>• 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.</li> </ul>
xml	<p>Attributes:</p> <ul style="list-style-type: none"> <li>• name (Required) - the unique query identifier.</li> <li>• expressionPath - Xpath expression</li> </ul>
url	<p>Attributes:</p> <ul style="list-style-type: none"> <li>• method - either GET or POST</li> <li>• realm - authentication name</li> <li>• username- valid username</li> <li>• password - valid password</li> </ul>

Element	Attributes/Description
link	<p>Attributes:</p> <ul style="list-style-type: none"> <li>• parentQuery - specify the parent query name.</li> <li>• parentColumn - specify the parent column name.</li> <li>• childQuery - specify the child query name.</li> <li>• childColumn - specify the child column name.</li> <li>• condition - the SQL operator that defines the relationship between the parent column and the child column. The following values for condition are supported: =, &lt;, &lt;=, &gt;, &gt;=</li> </ul>
dataTrigger	<p>Attributes:</p> <ul style="list-style-type: none"> <li>• name (Required) - the event name to fire this trigger</li> <li>• source (Required) - the PL/SQL &lt;package name&gt;.&lt;function name&gt;</li> </ul>
dataStructure	<p>(Required for multiple queries) Defines the structure of the output XML. Consists of &lt;group&gt; and &lt;element&gt;elements to specify the structure. This section is optional for single queries; if not specified, the data engine will generate flat XML.</p>
group	<p>Consists of one or more &lt;element&gt; elements and sub &lt;group&gt; elements.</p> <p>Attributes:</p> <ul style="list-style-type: none"> <li>• name (Required) - the XML tag name to be assigned to the group.</li> <li>• source (Required) - the unique query identifier for the corresponding sqlstatement from which the group's elements will be derived.</li> <li>• groupFilter - the filter to apply to the output data group set. Define the filter as: &lt;package name&gt;.&lt;function name&gt;.</li> </ul> <p><b>Note:</b> Applying a filter has performance impact. Do not use this functionality unless necessary. When possible, filter data using a WHERE clause in your query.</p>

Element	Attributes/Description
element (Required)	Attributes: <ul style="list-style-type: none"> <li>name - the tag name to assign to the element in the XML data output.</li> <li>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>.</li> <li>function - supported functions are: SUM(), COUNT(), AVG(), MIN(), MAX()</li> </ul>

## 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.

Attribute Name	Description
name	(Required) Enter the data template name.
description	(Optional) Enter a description of this data template.
version	(Required) Enter a version number for this data template.
defaultPackage	This attribute is required if your data template contains lexical references or any other calls to PL/SQL.
dataSourceRef	The default data source reference for the entire data template. Required only when performing a distributed query across multiple data sources. See Distributed Queries, page 5-32.

### Properties Section

Use the `<properties>` section to set properties to affect the XML output and data engine execution.

Example:

```
<properties>
  <property name="include_parameters" value="false" />
  <property name="include_null_Element" value="false" />
  <property name="include_rowsettag" value="false" />
  <property name="scalable_mode" value="on" />
</properties>
```

The following table shows the supported properties:

---

<b>Property Name</b>	<b>Description</b>
include_parameters	Indicates whether to include parameters in the output.  Valid values are: <ul style="list-style-type: none"><li>• True (default)</li><li>• False</li></ul>
include_null_Element	Indicates whether to remove or keep the null elements in the output.  Valid values are: <ul style="list-style-type: none"><li>• True (default)</li><li>• False</li></ul>
xml_tag_case	Allows you to set the case for the output XML element names.  Valid values are: <ul style="list-style-type: none"><li>• upper (default)</li><li>• lower</li><li>• as_are (The case will follow the definition in the dataStructure section.)</li></ul>

---

Property Name	Description
db_fetch_size	<p>Sets the number of rows fetched at a time through the jdbc connection. The default value is 500.</p> <p><b>Important:</b> For large queries with many columns, set the value to 20. Otherwise the memory footprint is significantly increased.</p>
scalable_mode	<p>Sets the data engine to execute in scalable mode. This is required when processing a large volume of data.</p> <p>Valid values:</p> <ul style="list-style-type: none"> <li>• on</li> <li>• off (default)</li> </ul>
include_rowsettag	<p>Allows you to include or exclude the Rowset Tag from the output.</p> <p>Valid values:</p> <ul style="list-style-type: none"> <li>• true (default)</li> <li>• false</li> </ul>
debug_mode	<p>Turns debug mode on or off.</p> <p>Valid values:</p> <ul style="list-style-type: none"> <li>• on</li> <li>• off (default)</li> </ul>

## 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:

```
<parameters>
  <parameter name="dept" dataType="number" defaultValue="10"/>
</parameters>
```

---

Attribute Name	Description
<code>name</code>	<b>Required.</b> A keyword, unique within a given Data Template, that identifies the parameter.
<code>dataType</code>	Optional. Specify the parameter data type as "character", "date", or "number". Default value is "character".  For the "date" dataType, the following three formats (based on the canonical ISO date format) are supported: <ul style="list-style-type: none"><li>• YYYY-MM-DD (example: 1997-10-24)</li><li>• YYYY-MM-DD HH24:MI:SS (example: 1997-10-24 12:00:00)</li><li>• YYYY-MM-DDTHH24:MI:SS.FF3TZH:TZM</li></ul>
<code>defaultValue</code>	Optional. This value will be used for the parameter if no other value is supplied from the data at runtime.
<code>include_in_output</code>	Optional. Whether this parameter should appear in XML output or not. The valid values are "true" and "false".

---

## How to Pass Parameters

To pass parameters, (for example, to restrict the query), use bind variables in your query. For example:

Query:

```
SELECT * FROM EMP
WHERE deptno=:department
```

At runtime, the value of `department` is passed to the query:

```
SELECT * FROM EMP
WHERE deptno=10
```

## Data Query Section

The `<dataQuery>` section of the data template is required.

## Supported Column Types

The following column types are selectable:

- VARCHAR2, CHAR
- NUMBER
- DATE, TIMESTAMP
- BLOB/BFILE (conditionally supported)

BLOB image retrieval is supported in the following two cases:

- Using the SetSQL API (see SQL to XML Processor, page 5-30)
- In the data template when no Structure section is defined. The returned data must be flat XML.

The BLOB/BFILE must be an image. Images are retrieved into your results XML as base64 encoding. You can retrieve any image type that is supported in the RTF template (jpg, gif, or png). You must use specific syntax to render the retrieved image in your template. See *Rendering an Image Retrieved from BLOB Data, Oracle XML Publisher Report Designer's Guide*.

- CLOB (conditionally supported)

The CLOB must contain text or XML. Data cannot be escaped inside the CLOB column.

- XMLType (conditionally supported)

XMLType can be supported if it is converted to a CLOB using the `getClobVal()` method.

- REF CURSOR (conditionally supported)

A REF CURSOR is supported inside the SQL statement when only one results set is returned.

## How to Define SQL 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:

```

<dataQuery>
  <sqlStatement name="Q1">
    <![CDATA[SELECT DEPTNO,DNAME,LOC from dept]]>
  </sqlStatement>
</dataQuery>

```

Attribute Name	Description
name	A unique identifying name for the query. Note that this name will be referred to throughout the data template.
dataSourceRef	(For E-Business Suite implementations only, not applicable for XML Publisher Enterprise). 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.

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.

## How to Define an XML Data Source

Place the `<xml>` element between the open and close `dataQuery` tags. The `<xml>` element has the related attributes: `name`, which is a unique identifier; and `expressionPath`, which can be used to link the SQL query and the XML data. Linking the SQL query and XML data enables you to leverage capabilities such as aggregation and summarization.

Example:

```
<xml name="empxml" expressionPath="//ROW[DEPTNO=$DEPTNO]">
<url method="GET" realm="" username=""
password="">file:///d:/dttest/employee.xml</url>
</xml>
```

## 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:

```
&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 <code>&lt;sqlstatement&gt;</code> element. See <i>How to Define Queries</i> , page 5-11.
parentColumn	Specify the parent column name.
childQuery	Specify the child query name. This must be the name that you assigned to the corresponding <code>&lt;sqlstatement&gt;</code> element. See <i>How to Define Queries</i> , page 5-11.
childColumn	Specify the child column name.
condition	The SQL operator that defines the relationship between the parent column and the child column. The following values for condition are supported: =, <, <=, >, >=

---

## 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:

```
<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:

```
<dataStructure>
  <group name="G_DEPT" source="Q1" ">
    <element name="DEPT_NUMBER" value="DEPTNO" />
    <element name="DEPT_NAME" value="DNAME"/>
    <group name="G_EMP" source="Q2">
      <element name="EMPLOYEE_NUMBER" value="EMPNO" />
      <element name="NAME" value="ENAME"/>
      <element name="JOB" value="JOB" />
    </group>
  </group>
</dataStructure>
```

The following table lists the attributes for the `<group>` element tag:

Attribute Name	Description
name	Specify any unique name for the group. This name will be used as the output XML tag name for the group.
source	The name of the query that provides the source data for the group. The <code>source</code> must come from the <code>name</code> attribute of the <code>&lt;sqlStatement&gt;</code> element.

The following table lists the attributes for the `<element>` element tag:

Attribute Name	Description
name	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.
value	The name of the column that provides the source data for the element (from your query).

## 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:

```
function G_EMPFilter return boolean is
begin
  if sal < 1000 then
    return (FALSE);
  else
    return (TRUE);
  end if;
end;
```

An example of the group filter in your data template definition would be:

```
<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()" />
```

## 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:

Attribute	Description
name	The XML tag name to be used in the XML data output.
source	The name of the column that contains the data on which the summary calculation is to be performed. The source column remains unchanged.
function	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().

The break group determines when to reset the value of the summary column. For example:

```

<group name="G_DEPT" source="Q1">
  <element name="DEPT_NUMBER" value="DEPTNO" />
  <element name="DEPTSAL" value="G_EMP.SALARY" function="SUM()" />
  <group name="G_EMP" source="Q2">
    <element name="EMPLOYEE_NUMBER" value="EMPNO" />
    <element name="NAME" value="ENAME"/>
    <element name="JOB" value="JOB" />
    <element name="SALARY" value="SAL"/>
  </group>
</group>

```

## Flexfield Support

**Note:** This section applies to data templates written to query the Oracle Applications database.

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 type="oracle.apps.fnd.flex.kff..."
      name="<Name of the lexical>"
      comment="<comment>"
    />
  </lexicals>

  <dataQuery>
    <sqlStatement ...

      SELECT &FLEX_SELECT flex_select_alias
      FROM some_table st, code_combination_table cct
      WHERE st.some_column = 'some_condition'
      AND &FLEX_WHERE
      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:

```
<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:

Attribute	Description
application_short_name	(Required) The application short name of the key flexfield. For example: SQLGL.
id_flex_code	(Required) the internal code of the key flexfield. For example: GL#
id_flex_num	(Required) Internal number of the key flexfield structure. For example: 101
segments	(Optional) Identifies for which segments this data is requested. Default value is "ALL". See the <i>Oracle Applications Developer's Guide</i> for syntax.

Attribute	Description
show_parent_segments	(Optional) Valid values are "Y" and "N". Default value is "Y". If a dependent segment is displayed, the parent segment is automatically displayed, even if it is not specified as displayed in the segments attribute.
metadata_type	(Required) Identifies what type of metadata is requested. Valid values are:  above_prompt - above prompt of segment(s).  left_prompt - left prompt of segment(s)

### 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,
&FLEX_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:

Attribute	Description
application_short_name	(Required) The application short name of the key flexfield. For example: SQLGL.
id_flex_code	(Required) the internal code of the key flexfield. For example: GL#
id_flex_num	(Conditionally required) Internal number of the key flexfield structure. For example: 101. Required if MULTIPLE_ID_FLEX_NUM is "N".
multiple_id_flex_num	(Optional) Indicates whether this lexical supports multiple structures or not. Valid values are "Y" and "N". Default is "N". If set to "Y", 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.
code_combination_table_alias	(Optional) Segment column names will be prepended with this alias.
segments	(Optional) Identifies for which segments this data is requested. Default value is "ALL". See the <i>Oracle Applications Developer's Guide</i> for syntax.
show_parent_segments	(Optional) Valid values are "Y" and "N". Default value is "Y". If a dependent segment is displayed, the parent segment is automatically displayed, even if it is not specified as displayed in the <code>segments</code> attribute.

Attribute	Description
output_type	<p>(Required) Indicates what kind of output should be used as the reported value. Valid values are:</p> <p>value - segment value as it is displayed to user.</p> <p>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.</p>
description	Segment value's description up to the description size defined in the segment definition.
full_description	Segment value's description (full size).
security	Returns Y if the current combination is secured against the current user, N otherwise.

### 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:

```

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:

Attribute	Description
application_short_name	(Required) The application short name of the key flexfield. For example: SQLGL.
id_flex_code	(Required) the internal code of the key flexfield. For example: GL#
id_flex_num	(Conditionally required) Internal number of the key flexfield structure. For example: 101. Required if MULTIPLE_ID_FLEX_NUM is "N".
code_combination_table_alias	(Optional) Segment column names will be prepended with this alias.
segments	(Optional) Identifies for which segments this data is requested. Default value is "ALL". See the <i>Oracle Applications Developer's Guide</i> for syntax.
operator	(Required) Valid values are: =, <, >, <=, >=, !=, <>,   , BETWEEN, LIKE
operand1	(Required) Values to be used on the right side of the conditional operator.

Attribute	Description
operand2	(Optional) High value for the BETWEEN operator.
full_description	Segment value's description (full size).
security	Returns Y if the current combination is secured against the current user, N otherwise.

### 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"
    operator="="
    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:

Attribute	Description
application_short_name	(Required) The application short name of the key flexfield. For example: SQLGL.
id_flex_code	(Required) the internal code of the key flexfield. For example: GL#
id_flex_num	(Conditionally required) Internal number of the key flexfield structure. For example: 101. Required if MULTIPLE_ID_FLEX_NUM is "N".
multiple_id_flex_num	(Optional) Indicates whether this lexical supports multiple structures or not. Valid values are "Y" and "N". Default is "N". If set to "Y", 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.
code_combination_table_alias	(Optional) Segment column names will be prepended with this alias.
segments	(Optional) Identifies for which segments this data is requested. Default value is "ALL". See the <i>Oracle Applications Developer's Guide</i> for syntax.
show_parent_segments	(Optional) Valid values are "Y" and "N". Default value is "Y". If a dependent segment is displayed, the parent segment is automatically displayed, even if it is not specified as displayed in the <code>segments</code> attribute.

## 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>
  <lexical
    type="oracle.apps.fnd.flex.kff.order_by"
    name="FLEX_ORDER_BY_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"
    show_parent_segments="Y"/>
  <lexical
    type="oracle.apps.fnd.flex.kff.order_by"
    name="FLEX_ORDER_BY_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"
    show_parent_segments="N"/>
</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 2-2. You will upload your data template XML file to the Template Manager.

## Calling a Data Template from 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 2-2.
- 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*.

## Calling a Data Template from the 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.

### 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 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>
  </dataQuery>
  <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 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>
    </group>
  </dataStructure>
</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 arraylist 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. Your 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

```
//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 SupplID 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.

### 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*, page 5-13.

#### 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`.

```

<?xml version="1.0" encoding="WINDOWS-1252" ?>
<dataTemplate name="data" description="DistQ" dataSourceRef="ODB"
  version="1.0"> <!-- Note the dataSourceRef attribute definition -->
  <parameters>
    <parameter name="DeptID" dataType="number" defaultValue="10" />
  </parameters>
  <dataQuery>
    <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
      ]]><!-- Notice this query is using the :DepartmentID bind variable
        from Q1 -->
    </sqlStatement>
  </dataQuery>
  <dataStructure><!-- No change in the data structure just referencing Q1
and Q2-->
    <group name="G_DEPT" source="Q1">
      <element name="DEPT_NUMBER" value="DepartmentID" />
      <element name="Name" value="Name" />
    </group>
    <group name="G_EMP" source="Q2">
      <element name="EMPLOYEE_NUMBER" value="EmployeeID" />
      <element name="Title" value="Title" />
      <element name="BirthDate" value="BirthDate" />
    </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/DistrubutedQuery.xml");

//Get jdbc connections to the distrubuted databases we wish to
//execute the template against. Assuming you have functions
//to retrieve these
    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/DistrubutedQuery_data.xml");
//Process the data template
    dp.processData();
}
catch (Exception e)
{
}
}

```

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 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 version="1.0" encoding="WINDOWS-1252" ?>- The template is named,
an optional description
- can be provided and the default package, if any, is identified:
<dataTemplate name="Employee Listing" description="List of
Employees" dataSourceRef="ORCL_DB1" defaultPackage="employee"
version="1.0">
  <parameters>- Defines a single parameter for the Department Number
- with default of 20:
    <parameter name="p_DEPTNO" dataType="character"
      defaultValue="20"/>
  </parameters>
  <dataQuery>
    <sqlStatement name="Q1">- This extracts the department
information based on a
- where clause from a pl/sql package:
      <![CDATA[SELECT DEPTNO,DNAME,LOC from dept
        where &pwhereclause
        order by deptno]]>
    </sqlStatement>
    <sqlStatement name="Q2">- This second query extracts the employee
data and joins to
- the parent query using a bind variable, :DEPTNO
      <![CDATA[SELECT EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,nvl
        (COMM,0) COMM
        from EMP
        WHERE DEPTNO = :DEPTNO ]]>
    </sqlStatement>
  </dataQuery>- A call is made to a before fetch trigger to set the

- where clause variable in the department query, &pwhereclause:

  <dataTrigger name="beforeReport"
    source="employee.beforeReportTrigger"/>
  <dataStructure>- The following section specifies the XML
hierarchy
- for the returning data:
    <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)

      <element name="DEPT_NUMBER" value="DEPTNO" />
      <element name="DEPT_NAME" value="DNAME"/>- This creates a
summary total at the department level based
- on the salaries at the employee level for each department:

      <element name="DEPTSAL" value="G_EMP.SALARY"
        function="SUM()"/>
    </group>
    <element name="LOCATION" value="LOC" />
    <group name="G_EMP" source="Q2">
      <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>

```

</dataTemplate>

### The PL/SQL Package:

```
- This is the package specification, it declares the global
- variables and functions contained therein
function BeforeReportTrigger return boolean;
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
FUNCTION BeforeReportTrigger return boolean is
begin
  IF (p_DEPTNO=10) THEN
    pwhereclause := 'DEPTNO =10';
  elsif (p_DEPTNO=20) THEN
    pwhereclause:='DEPTNO =20';
  elsif (p_DEPTNO=30) THEN
    pwhereclause:='DEPTNO =30';
  elsif (p_DEPTNO=40) THEN
    pwhereclause:='DEPTNO =20';
  else
    pwhereclause:='1=1';
  end if;
end;
RETURN(TRUE);
- 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 version="1.0" encoding="UTF-8" ?>
<dataTemplate name="GLRGNJ" dataSourceRef="ORA_EBS"
  defaultPackage="GLRGNJ" version="1.0">
  <parameters>- Parameter declaration, these will be populated at
runtime.
    <parameter name="P_CONC_REQUEST_ID" dataType = "number"
      defaultValue="0"></parameter>
    <parameter name="P_JE_SOURCE_NAME" dataType="character">
</parameter>
    <parameter name="P_SET_OF_BOOKS_ID" dataType="character"
      defaultValue="1"></parameter>
    <parameter name="P_PERIOD_NAME" dataType="character">Dec-97
</parameter>
    <parameter name="P_BATCH_NAME" dataType="character"></parameter>
    <parameter name="P_POSTING_STATUS" dataType="character"
      defaultValue="P"></parameter>
    <parameter name="P_CURRENCY_CODE" dataType="character"
      defaultValue="USD"></parameter>
    <parameter name="P_START_DATE" dataType = "date"></parameter>
    <parameter name="P_END_DATE" dataType = "date"></parameter>
    <parameter name="P_PAGESIZE" dataType = "number"
      defaultValue="180"></parameter>
    <parameter name="P_KIND" dataType = "character"
      defaultValue="L"></parameter>
  </parameters>
  <lexicals>- 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
<lexical type ="oracle.apps.fnd.flex.kff.select"
  name ="FLEXDATA_DSP"
  application_short_name="SQLGL"
  id_flex_code="GL#"
  id_flex_num=":STRUCT_NUM"
  multiple_id_flex_num="N"
  code_combination_table_alias="CC"
  segments="ALL"
  show_parent_segments="Y"
  output_type="VALUE" />- The second will return 'Y' if the current
combination is
- secured against the current user, 'N' otherwise
<lexical type ="oracle.apps.fnd.flex.kff.select"
  name ="FLEXDATA_SECURE"
  application_short_name="SQLGL"
  id_flex_code="GL#"
  id_flex_num=":STRUCT_NUM"
  multiple_id_flex_num="N"
  code_combination_table_alias="CC"
  segments="ALL"
  show_parent_segments="Y"
  output_type="SECURITY" />
</lexicals>
<dataQuery>
<sqlStatement name="Q_MAIN">
<![CDATA[
SELECT
S.user_je_source_name           Source,
B.name                          Batch_Name,
B.default_effective_date       Batch_Eff_date,
B.posted_date                   Batch_Posted_Date,

```

```

B.je_batch_id Batch_Id,
B.running_total_accounted_dr B_TOT_DR,
B.running_total_accounted_cr B_TOT_CR,
D.je_header_id Header_id,
D.name Header_Name,
C.user_je_category_name Category,
D.running_total_accounted_dr H_TOT_DR,
D.running_total_accounted_cr H_TOT_CR,
J.je_line_num Je_Line_Num,
decode(nvl(CC.code_combination_id, -1), -1, 'A', null)
FLEXDATA_H,
J.effective_date Line_Effective_Date,
J.description Line_Description,
J.accounted_dr Line_Acc_Dr,
J.accounted_cr Line_Acc_Cr,
D.currency_code Currency_Code,
D.external_reference Header_Reference,
&POSTING_STATUS_SELECT Recerence1_4,
nvl(J.stat_amount,0) Line_Stat_Amount,
GLL.description Batch_Type,
B.actual_flag Actual_Flag,
GLL2.meaning Journal_Type,
SOB.consolidation_sob_flag Cons_Sob_Flag,
&FLEXDATA_DSP FLEXDATA_DSP,
&FLEXDATA_SECURE FLEXDATA_SECURE
FROM gl_lookups GLL, gl_je_sources S, gl_je_categories C,
gl_je_lines J, gl_code_combinations CC, gl_je_headers D,
gl_je_batches B, gl_lookups GLL2, gl_sets_of_books SOB
WHERE GLL.lookup_code = B.actual_flag
AND GLL.lookup_type = 'BATCH_TYPE'
AND GLL2.lookup_type = 'AB_JOURNAL_TYPE'
AND GLL2.lookup_code = B.average_journal_flag
AND SOB.set_of_books_id = :P_SET_OF_BOOKS_ID
AND S.je_source_name = D.je_source
AND C.je_category_name = D.je_category
AND J.code_combination_id = CC.code_combination_id(+)
AND J.je_header_id = D.je_header_id
AND &CURRENCY_WHERE
AND D.je_source = NVL(:P_JE_SOURCE_NAME, D.je_source)
AND D.je_batch_id = B.je_batch_id
AND &POSTING_STATUS_WHERE
AND B.name = NVL(:P_BATCH_NAME, B.name)
AND &PERIOD_WHERE
AND B.set_of_books_id = :P_SET_OF_BOOKS_ID
ORDER BY S.user_je_source_name,
B.actual_flag,
B.name,
B.default_effective_date,
D.name,
J.je_line_num
]]>
</sqlStatement>

```

</dataQuery>- **The original report had an AfterParameter  
- and Before report triggers**

```

<dataTrigger name="afterParameterFormTrigger"
source="GLRGNJ.afterpform"/>
<dataTrigger name="beforeReportTrigger"
source="GLRGNJ.beforereport"/>

```

<dataStructure>- **A very complex XML hierarchy can be built with summary  
- columns referring to lower level elements**

```

<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 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"/>
    <element name="B_TOTAL_STAT" function="sum" dataType="number"
      value="G_HEADERS.H_Total_Stat"/>
    <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"/>
      <element name="H_Total_Stat" function="sum" dataType="number"
        value="G_LINES.Line_Stat_Amount"/>
    </group>
  </group>
  <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"/>
  </group>

```

```

<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>
</group>
<element name="R_TOT_DR"  function="sum"  dataType="number"
  value="G_SOURCE.SOU_SUM_ACC_DR"/>
<element name="R_TOT_CR"  function="sum"  dataType="number"
  value="G_SOURCE.SOU_SUM_ACC_CR"/>
<element name="R_TOT_STAT_AMT"  function="sum"  dataType="number"
  value="G_SOURCE.SOU_SUM_STAT_AMT"/>
<element name="JE_SOURCE_DSP"  function="first"  dataType="number"
  value="G_SOURCE.Source"/>
<element name="REP_BATCH_ID"  function="first"  dataType="number"
  value="G_BATCHES.Batch_Id"/>
<element name="C_DATEFORMAT"  dataType="varchar2"
  value="C_DATEFORMAT"/>
</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>

```

## Employee XML Datasource Data Template

This data template combines data that exists in a table called "dept" with data from an xml file called "employee.xml". It follows the same format as the Employee data template but the employee data comes from an xml file instead of from the emp table.

```

<?xml version="1.0" encoding="WINDOWS-1252" ?>
<dataTemplate name="Employee Listing" description="List of Employees" version="1.0">
  <parameters>- Defines a single parameter for the Department Number
    - with default of 20:
    <parameter name="p_DEPTNO" dataType="character"
      defaultValue="20"/>
  </parameters>
  <dataQuery>
    <sqlStatement name="Q1">
      <![CDATA[SELECT DEPTNO,DNAME,LOC from dept
        order by deptno]]>
    </sqlStatement>
    <xml name="empxml" expressionPath="//ROW[DEPTNO=$DEPTNO]">- Defines
name
    - and link to DEPTNO in Q1
    <url method="GET" realm="" username="" password=""
      file:///d:/dttest/employee.xml</url>- Defines url for xml data
    </xml>
  </dataQuery>-
    <dataStructure>- The following section specifies the XML hierarchy-
for the returning data:
    <group name="G_DEPT" source="Q1"
      <element name="DEPT_NUMBER" value="DEPTNO" />
      <element name="DEPT_NAME" value="DNAME"/>
    - This creates a summary total at the department level based
    - on the salaries at the employee level for each department:

    <element name="DEPTSAL" value="G_EMP.SALARY"
      function="SUM()"/>
    <element name="LOCATION" value="LOC" />
    <group name="G_EMP" source="empxml">
      <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>
  </group>
</dataStructure>
</dataTemplate>

```

---

## Calling XML Publisher APIs

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
- Advanced Barcode Font Formatting Implementation

### 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 Oracle *MetaLink* document Note 422508.1, "About Oracle XML Publisher Release 5.6.3."

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.

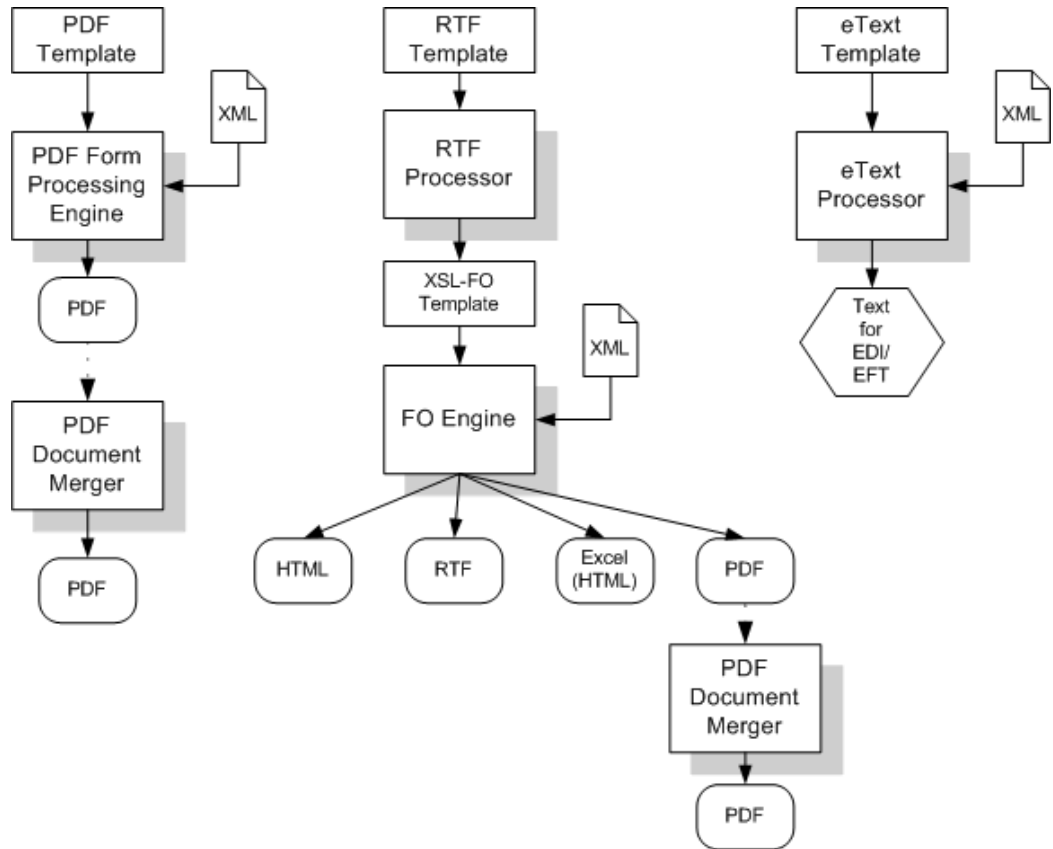
This section assumes the reader is familiar with Java programming, XML, and XSL technologies. For the Applications layer, it is assumed the reader is familiar with the Template Manager.

## 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 (.xtn) 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

```
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 XML
Data
    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"?>
<xfdf xmlns="http://ns.adobe.com/xfdf/" xml:space="preserve">
<fields>
  <field name="TITLE">
    <value>Purchase Order</value>
  </field>
  <field name="SUPPLIER_TITLE">
    <value>Supplie</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:

```

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<fields>
<xsl:apply-templates/>
</fields>
</xsl:template>
  <!-- Count how many ROWs(rows) are in the source XML file. -->
  <xsl:variable name="cnt" select="count(//row//ROW)" />
  <!-- Try to match ROW (or row) element.
  <xsl:template match="ROW/*|row/*">
    <field>
      <!-- Set "name" attribute in "field" element. -->
      <xsl:attribute name="name">
        <!-- Set the name of the current element (column name) as a
value of the current name attribute. -->
        <xsl:value-of select="name(.)" />
        <!-- Add the number at the end of the name attribute value
if more than 1 rows found in the source XML file.-->
        <xsl:if test="$cnt > 1">
          <xsl:number count="ROW|row" level="single" format="1"/>
        </xsl:if>
      </xsl:attribute>
      <value>
        <!--Set the text data set in the current column data as a
text of the "value" element. -->
        <xsl:value-of select="." />
      </value>
    </field>
  </xsl:template>
</xsl:stylesheet>

```

You can then use the `XFDFObj` to convert XML to the XFDf format using an XSLT as follows:

### Example

```

import java.io.*;
import oracle.apps.xdo.template.pdf.xfdf.XFDFObj;
.
.
.
XFDFObj xfdfObj = new XFDFObj();

xfdfObj .setStylesheet(new BufferedInputStream(new
FileInputStream(xslPath))); // XSL file name
xfdfObj .appendXML( new File(xmlPath1)); // XML data file name
xfdfObj .appendXML( new File(xmlPath2)); // XML data file name

System.out.print(xfdfObj .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 inputStream outputStream
        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. For a list of supported XSL-FO elements, see Supported XSL-FO Elements, *Oracle XML Publisher Report Designer's Guide*.

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.

### 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.

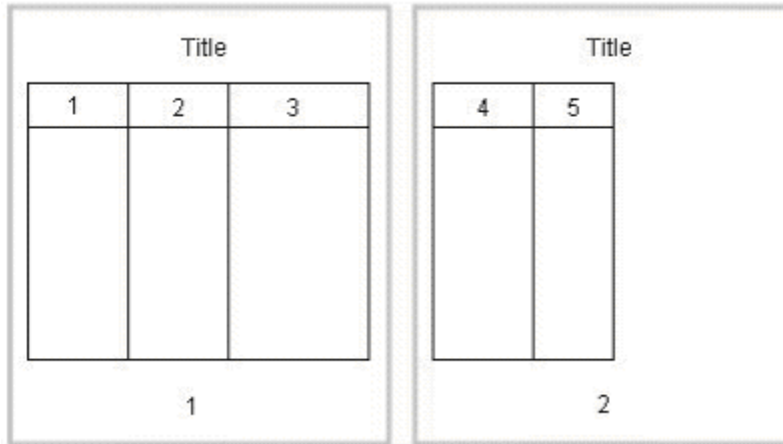
The following figure shows a table that is too wide to display on one page:

The diagram shows a table with a title 'Title' centered above it and a page number 'Page Number' centered below it. The table has five columns labeled 1, 2, 3, 4, and 5. The first three columns are wider than the last two. The table is shown within a rectangular frame that is wider than the table itself, indicating it is too wide to fit on one page.

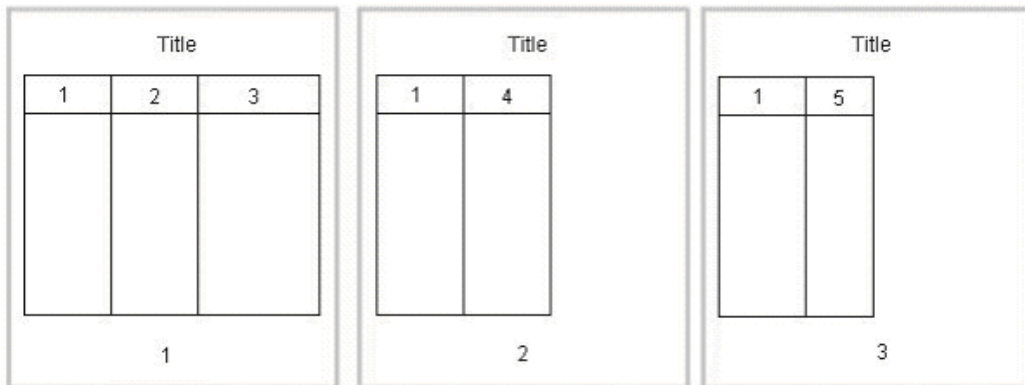
Title				
1	2	3	4	5

Page Number

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)

### 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 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

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.process();
        // 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 6-18.

### Creating XSL-FO from an XML File and an XSL File

Input:

- XML file
- XSL file

Output:

- XSL-FO (InputStream)

### Example

```
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

```
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

```
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

```
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

```
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:

```
float[ ] rct = {LowerLeft X, LowerLeft Y, UpperRight X,
UpperRight Y}
```

For example:

```
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:

```
float[ ] rct = {100f, 100f, -1f, -1f}
```

Input:

- PDF Documents (InputStream)
- Image File (InputStream)

Output:

- PDF Document (OutputStream)

### Example

```
import java.io.*;
import oracle.apps.xdo.common.pdf.util.PDFDocMerger;
.
.
.
public boolean mergeDocs(InputStream inputStreams, OutputStream
outputStream, String imagePath)
{
    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.

### 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.

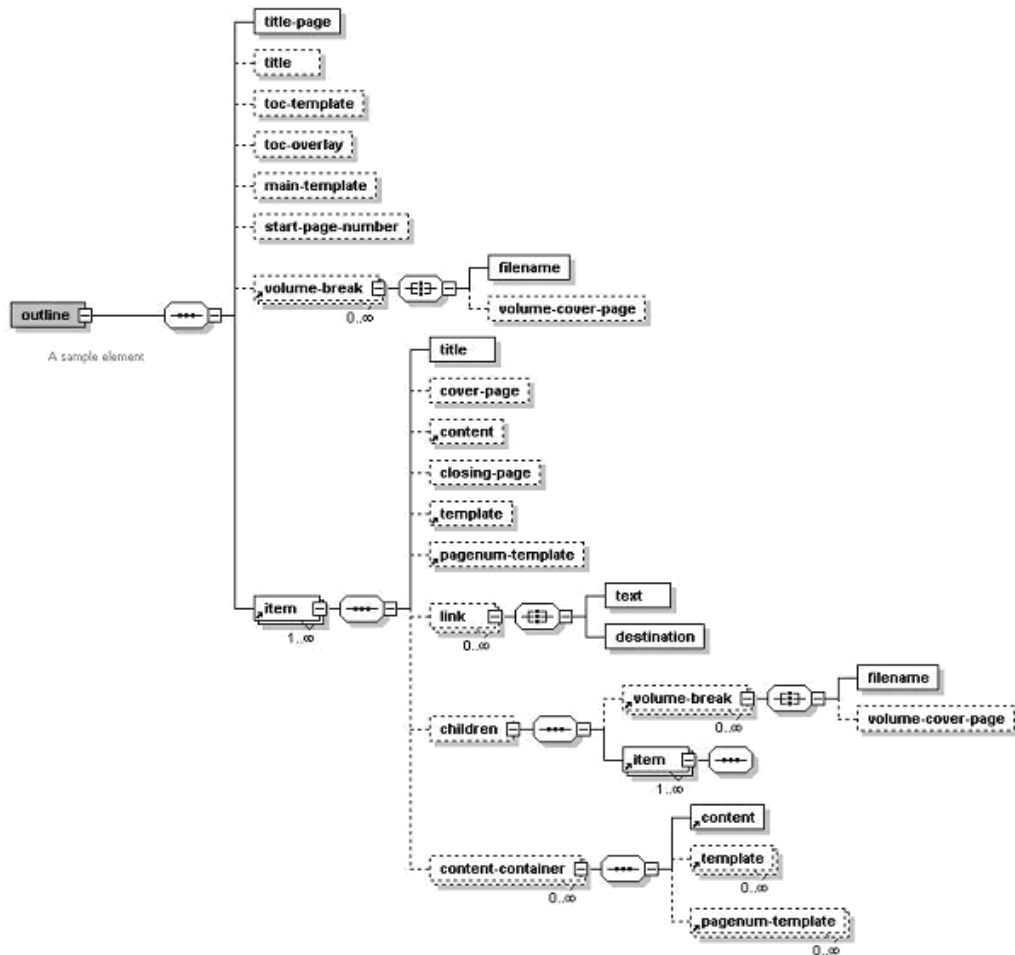
The templates can be rotated (at right angles) or be made transparent. A PDF template can also be specified at the book level, enabling the ability to specify global page numbering, or other content such as backgrounds and watermarks. A title page can also be passed in as a parameter, as well as cover 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 the following XML Publisher protocol:

- "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:

```
java oracle.apps.template.pdf.book.PDFBookBinder [-debug <true or false>] [-tmp <temp dir>] -xml <input xml> -pdf <output pdf>
```

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.

-tmp <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:

```
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

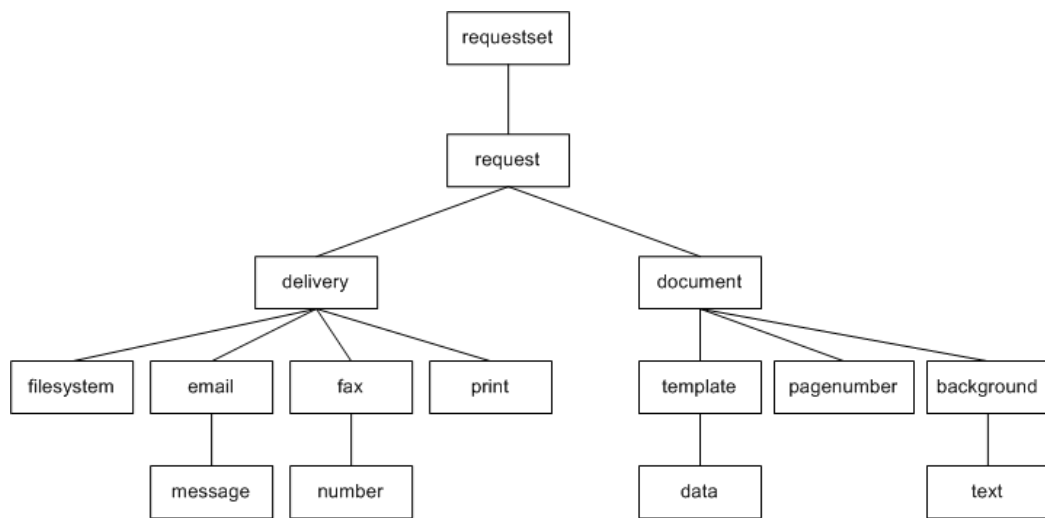
The Document Processor XML file has the following element hierarchy:

```

Requestset
  request
    delivery
      filesystem
      print
      fax
        number
      email
        message
    document
      background
      text
      pagenumber
      template
      data

```

This hierarchy is displayed in the following illustration:



The following table describes each of the elements:

Element	Attributes	Description
requestset	xmlns version	Root element must contain [ xmlns:xapi="http://xmlns.oracle.com/oxp/xapi/" ] block  The version is not required, but defaults to "1.0".
request	N/A	Element that contains the data and template processing definitions.

Element	Attributes	Description
delivery	N/A	Defines where the generated output is sent.
document	output-type	Specify one output that can have several template elements. The output-type attribute is optional. Valid values are:  pdf (Default)  rtf  html  excel  text
filesystem	output	Specify this element to save the output to the file system. Define the directory path in the output attribute.
print	<ul style="list-style-type: none"> <li>printer</li> <li>server-alias</li> </ul>	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: "ipp://myprintserver:631/printers/printername"
fax	<ul style="list-style-type: none"> <li>server</li> <li>server-alias</li> </ul>	Specify a URI in the server attribute, for example: "ipp://myfaxserver1:631/printers/myfaxmachine"
number		The number element can occur multiple times to list multiple fax numbers. Each element occurrence must contain only one number.

Element	Attributes	Description
email	<ul style="list-style-type: none"> <li>• server</li> <li>• port</li> <li>• from</li> <li>• reply-to</li> <li>• server-alias</li> </ul>	<p>Specify the outgoing mail server (SMTP) in the <code>server</code> attribute.</p> <p>Specify the mail server port in the <code>port</code> attribute.</p>
message	<ul style="list-style-type: none"> <li>• to</li> <li>• cc</li> <li>• bcc</li> <li>• attachment</li> <li>• subject</li> </ul>	<p>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.</p> <p>You can specify multiple e-mail addresses in the <code>to</code>, <code>cc</code> and <code>bcc</code> attributes separated by a comma.</p> <p>The <code>attachment</code> value is either <code>true</code> or <code>false</code> (default). If <code>attachment</code> is <code>true</code>, then a generated document will be attached when the e-mail is sent.</p> <p>The <code>subject</code> attribute is optional.</p>
background	where	<p>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.</p>

Element	Attributes	Description
text	<ul style="list-style-type: none"> <li>title</li> <li>default</li> </ul>	<p>Specify the watermark text in the <code>title</code> value.</p> <p>A default value of "yes" automatically draws the watermark with forward slash type. The default value is yes.</p>
pagenumber	<ul style="list-style-type: none"> <li>initial-page-index</li> <li>initial-value</li> <li>x-pos</li> <li>y-pos</li> </ul>	<p>The <code>initial-page-index</code> default value is 0.</p> <p>The <code>initial-value</code> default value is 1.</p> <p>"Helvetica" is used for the page number font.</p> <p>The <code>x-pos</code> provides lower left x position.</p> <p>The <code>y-pos</code> provides lower left y position.</p>
template	<ul style="list-style-type: none"> <li>locale</li> <li>location</li> <li>type</li> </ul>	<p>Contains template information.</p> <p>Valid values for the <code>type</code> attribute are</p> <p>pdf</p> <p>rtf</p> <p>xsl-fo</p> <p>etext</p> <p>The default value is "pdf".</p>

Element	Attributes	Description
data	location	<p>Define the <code>location</code> attribute to specify the location of the data, or attach the actual XML data with subelements. The default value of <code>location</code> is "inline". If the <code>location</code> points to either an XML file or a URL, then the data should contain an XML declaration with the proper encoding.</p> <p>If the <code>location</code> attribute is not specified, the <code>data</code> element should contain the subelements for the actual data. This must not include an XML declaration.</p>

## 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>data1</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 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:

```

<?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\out4.pdf"/>
    </xapi:delivery>

    <xapi:document output-type="pdf">
      <xapi:template type="pdf"
        location="http://your.server:9999/templates/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: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>

```

### 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 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:pagenumber initial-value="1" initial-page-index="1"
        x-pos="300" y-pos="20" />
      <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: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(dataFile,
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

```
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 6-31 and has its own method called to invoke it. The Document Processor XML 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 6-31).

## Control File

The control file takes the same format as the Document Processor XML, page 6-31 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 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="rgmamersmtp.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><!-- 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:email>
    </xapi:delivery>
    <xapi:document output-type="pdf" delivery="123">
      <xapi:template type="rtf" location="/usr/tmp/empGeneric.rtf">
      <xapi:template type="rtf" location="/usr/tmp/empDet.rtf"
filter="//EMPLOYEE[ENAME='SMITH']"><!-- Employees with the name
SMITH will have
the empDet template applied -->
      </xapi:template>
    </xapi:document>
  </xapi:request>
</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 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="gsmtp.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 be 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 version="1.0" encoding="UTF-8"?>
<DATA>
<DEPTS>
<DEPT>
  <DEPTNO>20</DEPTNO>
  <NAME>Accounting</NAME>
  <MANAGER_EMAIL>tdexter@mycomp.com</MANAGER_EMAIL>
  <EMPLOYEES>
    <EMPLOYEE>
      <EMPNO>7369</EMPNO>
      <ENAME>SMITH</ENAME>
      <JOB>CLERK</JOB>
      <MGR>7902</MGR>
      <HIREDATE>1980-12-17T00:00:00.000-08:00</HIREDATE>
      <SAL>800</SAL>
      <DEPTNO>20</DEPTNO>
      <EMAIL>jsmith@mycomp.com</EMAIL>
    </EMPLOYEE>
    <EMPLOYEE>
      <EMPNO>7566</EMPNO>
      <ENAME>JONES</ENAME>
      <JOB>MANAGER</JOB>
      <MGR>7839</MGR>
      <HIREDATE>1981-04-02T00:00:00.000-08:00</HIREDATE>
      <SAL>2975</SAL>
      <DEPTNO>20</DEPTNO>
      <EMAIL>jjones@mycomp.com</EMAIL>
    </EMPLOYEE>
  </EMPLOYEES>
</DEPT>
<DEPT>
  <DEPTNO>30</DEPTNO>
  <NAME>Sales</NAME>
  <MANAGER_EMAIL>dsmith@mycomp.com</MANAGER_EMAIL>
  <EMPLOYEES>
    <EMPLOYEE>
      <EMPNO>7788</EMPNO>
      <ENAME>SCOTT</ENAME>
      <JOB>ANALYST</JOB>
      <MGR>7566</MGR>
      <HIREDATE>1982-12-09T00:00:00.000-08:00</HIREDATE>
      <SAL>3000</SAL>
      <DEPTNO>20</DEPTNO>
      <EMAIL>jscott@mycomp.com</EMAIL>
    </EMPLOYEE>
    <EMPLOYEE>
      <EMPNO>7876</EMPNO>
      <ENAME>ADAMS</ENAME>
      <JOB>CLERK</JOB>
      <MGR>7788</MGR>
      <HIREDATE>1983-01-12T00:00:00.000-08:00</HIREDATE>
      <SAL>1100</SAL>
      <EMAIL>jadams@mycomp.com</EMAIL>
    </EMPLOYEE>
  </EMPLOYEES>
</DEPT>
</DEPTS>
</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><!--
Embed the employees name into the email message - >
</xapi:email>
  </xapi:delivery>
  <xapi:document output-type="pdf" delivery="123"><!--Specify the
delivery method id to be used - >
    <xapi:template type="rtf"
      location="\usr\empDet.rtf"></xapi:template>
    </xapi:document>
  </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.

```
<?xml version="1.0" encoding="UTF-8"?>
<xapi:requestset xmlns:xapi="http://xmlns.oracle.com/oxp/xapi" >
  <xapi:globalData location="stream">
  </xapi:globalData >
  <xapi:request select="/DATA/DEPTS/DEPT/EMPLOYEES/EMPLOYEE">
    <xapi:delivery>
      <xapi:email server="my.smtp.server" port=""
        from="xmlpserver@oracle.com"
        reply-to="reply@oracle.com">
        <xapi:message id="123" to="{EMAIL}" cc="" attachment="true"
          subject="Employee Details for {ENAME}"> Mr. {ENAME},
          Please review the attached document</xapi:message>
      </xapi:email>
      <xapi:print id="printer1"
        printer="ipp://ipgpc1:631/printers/printer1" copies="2" /><!-- Add an
id for this delivery method i.e. printer1 - > </xapi:delivery>
      <xapi:document output-type="pdf" delivery="printer1,123"><!--
Deliver to printer and email - > <xapi:template type="rtf"
location="/usr/empDetSmith.rtf"
      filter="//EMPLOYEE[ENAME='SMITH']"><!-- Specify template to be
used for employees called SMITH - >
      </xapi:template>
      <xapi:template type="rtf" location="/usr/empSummary.rtf"><!--
Default template to be used - >
      </xapi:template>
    </xapi:document>
  </xapi:request>
</xapi:requestset>
```

### 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 version="1.0" encoding="UTF-8"?>
<xapi:requestset xmlns:xapi="http://xmlns.oracle.com/oxp/xapi">
  <xapi:request select="/DATA/DEPTS/DEPT/EMPLOYEES/EMPLOYEE">
    <xapi:delivery>
      <xapi:email server="my.smtp.server" port=""
        from="xmlpserver@oracle.com" reply-to="">
        <xapi:message id="123" to="{EMAIL}" cc="{EMAIL}"
          attachment="true"
          subject="Employee Details for {ENAME}"> Mr. {ENAME},
          Please review the attached document</xapi:message>
        </xapi:email>
      </xapi:delivery>
      <xapi:document output-type="pdf" delivery="123">
        <xapi:template type="rtf"
          location="/usr/empDet.rtf"></xapi:template>
      </xapi:document>
    </xapi:request>
    <xapi:request select="/DATA/DEPTS/DEPT"><!-- Second request created to
burst the same dataset to the
manager based on the DEPT element -->
      <xapi:delivery>
        <xapi:email server="my.smtp.server" port=""
          from="xmlpserver@oracle.com" reply-to="">
        <xapi:message id="456" 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="456">
        <xapi:template type="rtf"
          location="\usr\deptSumm.rtf"></xapi:template>
      </xapi:document>
    </xapi:request>
  </xapi:requestset>
```

## 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 4-2 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, *Oracle XML Publisher Report Designer's Guide* 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();
    /* PDF Security control: */
    prop.put("pdf-security", "true");
    /* Permissions password: */
    prop.put("pdf-permissions-password", "abc");
    /* Encryption level: */
    prop.put("pdf-encryption-level", "0");
    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)

```

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

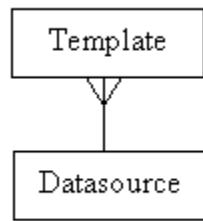
```
import oracle.apps.xdo.batch.DocumentProcessor;
.
.
.
public static void main(String[] args)
{
.
.
.
    try
    {
        // dataFile --- File path of the Document Processor XML
        // tempDir --- Temporary Directory path
        DocumentProcessor docProcessor = new DocumentProcessor(dataFile,
tempDir);
        Properties prop = new Properties();
        /* PDF Security control: */
        prop.put("pdf-security", "true");
        /* Permissions password: */
        prop.put("pdf-permissions-password", "abc");
        /* encryption level: */
        prop.put("pdf-encryption-level", "0");
        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  

```
((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

```
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()).getAppsContext()`  
where `am` is your `OApplicationModule`.
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 OApplicationModule Instance

Some methods require the `OApplicationModule` instance to communicate with the Applications database. Get the `OApplicationModule` instance in your code as follows:

1. If you are using the TemplateHelper in OA Framework, you already have an `OApplicationModule` instance

2. If you already have `AppsContext`, you can create the `OApplicationModule` instance by using  
`oracle.apps.fnd.framework.server.OApplicationModuleUtil`

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 `OApplicationModule`, 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(
    applicationContext,          // ApplicationContext
    "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(
    applicationContext,          // ApplicationContext
    "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(
    applicationContext,          // ApplicationContext
    "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(
    applicationContext,          // ApplicationContext
    "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

```
// 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.OUTPUT_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

```
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

```
TemplateHelper.processTemplate(appsContext, "OKC", "OKCTERMS", "en",  
"US", pData, props, pOutput);
```

3. Define XSL parameters in an RTF/FO template using the following syntax:

```
<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

```
// 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.  
                           // All properties will be passed to EFT/EDI  
    engine  
    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

```
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 2-7 for more information on assigning a default template file.

For example, the following table shows a sample of templates in the Template Manager repository:

Template File	ISO Language Code	ISO Territory Code	Default?
A	en	US	no
B	en	00	no
C	fr	FR	yes
D	ja	JP	no

The following table shows the template that will be returned if you pass the given ISO language/territory code combinations:

ISO Language Code	ISO Territory Code	Template Returned
en	US	A
en	GB	B
en	null	B
fr	FR	C
ja	JP	D
de	DE	C

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 xliiff 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 xliiff document will have empty elements.
- `uploadXLIFF()` - Uploads the translations for a template in xliiff format. The xliiff file must contain a valid target-language attribute.

## Advanced Barcode Font Formatting Implementation

For the advanced formatting to work in the template, you must provide a Java class with the appropriate methods to format the data at runtime. Many font vendors offer the code with their fonts to carry out the formatting; these must be incorporated as methods into a class that is available to the XML Publisher formatting libraries at runtime. There are some specific interfaces that you must provide in the class for the library to call the correct method for encoding.

**Note:** See Advanced Barcode Formatting, *Oracle XML Publisher Report Designer's Guide* for the setup required in the RTF template.

You must implement the following three methods in this class:

```
/**
 * Return a unique ID for this bacode encoder
 * @return the id as a string
 */
public String getVendorID();

/**
 * Return true if this encoder support a specific type of barcode
 * @param type the type of the barcode
 * @return true if supported
 */
public boolean isSupported(String type);

/**
 * Encode a barcode string by given a specific type
 * @param data the original data for the barcode
 * @param type the type of the barcode
 * @return the formatted data
 */
public String encode(String data, String type);
```

Place this class in the classpath for the middle tier JVM in which XML Publisher is running.

**Note:** For E-Business Suite users, the class must be placed in the classpath for the middle tier and any concurrent nodes that are present.

If in the register-barcode-vendor command the `barcode_vendor_id` is not provided, XML Publisher will call the `getVendorID()` and use the result of the method as the ID for the vendor.

The following is an example class that supports the code128 a, b and c encodings:

**Important:** The following code sample can be copied and pasted for use in your system. Note that due to publishing constraints you will need to correct line breaks and ensure that you delete quotes that display as

"smart quotes" and replace them with simple quotes.

### Example

```
package oracle.apps.xdo.template.rtf.util.barcode;  
  
import java.util.Hashtable;  
import java.lang.reflect.Method;  
import oracle.apps.xdo.template.rtf.util.XDOBarcodeEncoder;  
import oracle.apps.xdo.common.log.Logger;  
// This class name will be used in the register vendor  
// field in the template.  
  
public class BarcodeUtil implements XDOBarcodeEncoder  
// The class implements the XDOBarcodeEncoder interface  
{  
// This is the barcode vendor id that is used in the  
// register vendor field and format-barcode fields  
    public static final String BARCODE_VENDOR_ID = "XMLPBarVendor";  
// The hashtable is used to store references to  
// the encoding methods  
    public static final Hashtable ENCODERS = new Hashtable(10);  
// The BarcodeUtil class needs to be instantiated  
    public static final BarcodeUtil mUtility = new BarcodeUtil();  
// This is the main code that is executed in the class,  
// it is loading the methods for the encoding into the hashtable.  
// In this case we are loading the three code128 encoding  
// methods we have created.  
    static {  
        try {  
            Class[] clazz = new Class[] { "" .getClass() };  
  
            ENCODERS.put("code128a",mUtility.getClass().getMethod("code128a",  
clazz));  
            ENCODERS.put("code128b",mUtility.getClass().getMethod("code128b",  
clazz));  
            ENCODERS.put("code128c",mUtility.getClass().getMethod("code128c",  
clazz));  
        } catch (Exception e) {  
// This is using the XML Publisher logging class to push  
// errors to the XMLP log file.  
            Logger.log(e,5);  
        }  
    }  
}
```

```

// The getVendorID method is called from the template layer
// at runtime to ensure the correct encoding method are used
    public final String getVendorID()
    {
        return BARCODE_VENDOR_ID;
    }
//The isSupported method is called to ensure that the
// encoding method called from the template is actually
// present in this class.
// If not then XMLP will report this in the log.
    public final boolean isSupported(String s)
    {
        if(s != null)
            return ENCODERS.containsKey(s.trim().toLowerCase());
        else
            return false;
    }

// The encode method is called to then call the appropriate
// encoding method, in this example the code128a/b/c methods.

    public final String encode(String s, String s1)
    {
        if(s != null && s1 != null)
        {
            try
            {
                Method method =
(Method)ENCODERS.get(s1.trim().toLowerCase());
                if(method != null)
                    return (String)method.invoke(this, new Object[] {
                        s
                    });
                else
                    return s;
            }
            catch(Exception exception)
            {
                Logger.log(exception,5);
            }
            return s;
        } else
        {
            return s;
        }
    }

/** This is the complete method for Code128a */

    public static final String code128a( String DataToEncode )
    {
        char C128_Start = (char)203;
        char C128_Stop = (char)206;
        String Printable_string = "";
        char CurrentChar;
        int CurrentValue=0;
        int weightedTotal=0;
        int CheckDigitValue=0;
        char C128_CheckDigit='w';

        DataToEncode = DataToEncode.trim();

```

```

weightedTotal = ((int)C128_Start) - 100;
for( int i = 1; i <= DataToEncode.length(); i++ )
{
//get the value of each character
CurrentChar = DataToEncode.charAt(i-1);
if( ((int)CurrentChar) < 135 )
    CurrentValue = ((int)CurrentChar) - 32;
if( ((int)CurrentChar) > 134 )
    CurrentValue = ((int)CurrentChar) - 100;

CurrentValue = CurrentValue * i;
weightedTotal = weightedTotal + CurrentValue;
}
//divide the WeightedTotal by 103 and get the remainder, //this is
the CheckDigitValue
CheckDigitValue = weightedTotal % 103;
if( (CheckDigitValue < 95) && (CheckDigitValue > 0) )
    C128_CheckDigit = (char)(CheckDigitValue + 32);
if( CheckDigitValue > 94 )
    C128_CheckDigit = (char)(CheckDigitValue + 100);
if( CheckDigitValue == 0 ){
    C128_CheckDigit = (char)194;
}

Printable_string = C128_Start + DataToEncode + C128_CheckDigit +
C128_Stop + " ";
return Printable_string;
}

```

```

/** This is the complete method for Code128b ***/

public static final String code128b( String DataToEncode )
{
    char C128_Start = (char)204;
    char C128_Stop = (char)206;
    String Printable_string = "";
    char CurrentChar;
    int CurrentValue=0;
    int weightedTotal=0;
    int CheckDigitValue=0;
    char C128_CheckDigit='w';

    DataToEncode = DataToEncode.trim();
    weightedTotal = ((int)C128_Start) - 100;
    for( int i = 1; i <= DataToEncode.length(); i++ )
    {
//get the value of each character
        CurrentChar = DataToEncode.charAt(i-1);
        if( ((int)CurrentChar) < 135 )
            CurrentValue = ((int)CurrentChar) - 32;
        if( ((int)CurrentChar) > 134 )
            CurrentValue = ((int)CurrentChar) - 100;

        CurrentValue = CurrentValue * i;
        weightedTotal = weightedTotal + CurrentValue;
    }
//divide the WeightedTotal by 103 and get the remainder, //this is
the CheckDigitValue
    CheckDigitValue = weightedTotal % 103;
    if( (CheckDigitValue < 95) && (CheckDigitValue > 0) )
        C128_CheckDigit = (char)(CheckDigitValue + 32);
    if( CheckDigitValue > 94 )
        C128_CheckDigit = (char)(CheckDigitValue + 100);
    if( CheckDigitValue == 0 ){
        C128_CheckDigit = (char)194;
    }

    Printable_string = C128_Start + DataToEncode + C128_CheckDigit +
    C128_Stop + " ";
    return Printable_string;
}

/** This is the complete method for Code128c **/

public static final String code128c( String s )
{
    char C128_Start = (char)205;
    char C128_Stop = (char)206;
    String Printable_string = "";
    String DataToPrint = "";
    String OnlyCorrectData = "";
    int i=1;
    int CurrentChar=0;
    int CurrentValue=0;
    int weightedTotal=0;
    int CheckDigitValue=0;
    char C128_CheckDigit='w';
    DataToPrint = "";
    s = s.trim();

```

```

for(i = 1; i <= s.length(); i++ )
{
    //Add only numbers to OnlyCorrectData string
    CurrentChar = (int)s.charAt(i-1);
    if((CurrentChar < 58) && (CurrentChar > 47))
    {
        OnlyCorrectData = OnlyCorrectData + (char)s.charAt(i-1);
    }
    s = OnlyCorrectData;
    //Check for an even number of digits, add 0 if not even
    if( (s.length() % 2) == 1 )
    {
        s = "0" + s;
    }
    //<<<< Calculate Modulo 103 Check Digit and generate
    // DataToPrint >>>> //Set WeightedTotal to the Code 128 value of
// the start character
    weightedTotal = ((int)C128_Start) - 100;
    int WeightValue = 1;
    for( i = 1; i <= s.length(); i += 2 )
    {
        //Get the value of each number pair (ex: 5 and 6 = 5*10+6 =56) //And
assign the ASCII values to DataToPrint
        CurrentChar = (((int)s.charAt(i-1))-48)*10 + (((int)s.charAt(i))-48);
        if((CurrentChar < 95) && (CurrentChar > 0))
            DataToPrint = DataToPrint + (char)(CurrentChar + 32);
        if( CurrentChar > 94 )
            DataToPrint = DataToPrint + (char)(CurrentChar + 100);
        if( CurrentChar == 0 )
            DataToPrint = DataToPrint + (char)194;
        //multiply by the weighting character
        //add the values together to get the weighted total
        weightedTotal = weightedTotal + (CurrentChar * WeightValue);
        WeightValue = WeightValue + 1;
    }
    //divide the WeightedTotal by 103 and get the remainder, //this is
the CheckDigitValue
    CheckDigitValue = weightedTotal % 103;
    if((CheckDigitValue < 95) && (CheckDigitValue > 0))
        C128_CheckDigit = (char)(CheckDigitValue + 32);
    if( CheckDigitValue > 94 )
        C128_CheckDigit = (char)(CheckDigitValue + 100);
    if( CheckDigitValue == 0 ){
        C128_CheckDigit = (char)194;
    }
    Printable_string = C128_Start + DataToPrint + C128_CheckDigit +
    C128_Stop + "-";
    Logger.log(Printable_string,5);
    return Printable_string;
}
}

```

Once you create the class and place it in the correct classpath, your template creators can start using it to format the data for barcodes. You must give them the following information to include in the template commands:

- The class name and path.

In this example:

```
oracle.apps.xdo.template.rtf.util.barcoder.BarcodeUtil
```

- The barcode vendor ID you created.

In this example: XMLPBarVendor

- The available encoding methods.

In this example, code128a, code128b and code128c They can then use this information to successfully encode their data for barcode output.

They can then use this information to successfully encode their data for barcode output.

---

# Using the Delivery Manager APIs

## 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:

- 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 7-33.

The follow delivery channels are described in this document:

- E-mail
- Printer
- Fax
- WebDAV
- FTP
- Secure FTP
- HTTP
- AS2

## Delivering Documents via e-Mail

The following sample demonstrates delivery via E-mail:

## Example

```
// 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, "mysmtpost");
// set the sender email address
req.addProperty(DeliveryPropertyDefinitions.SMTP_FROM,
"myname@mydomain.com");
// set the destination email address
req.addProperty(
DeliveryPropertyDefinitions.SMTP_TO_RECIPIENTS,
"user1@mydomain.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_FILENAME,
"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:

Property	Description
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.

<b>Property</b>	<b>Description</b>
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.

Property	Description
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 (`oracle.apps.xdo.delivery.smtp.Attachment`) to attach multiple documents into one request. Sample usage is as follows:

## Example

```
:
:
// create Attachment instance
Attachment m = new Attachment();

// add PDF attachment
m.addAttachment(
    "/pdf_doc/invoice.pdf",    // file to deliver
    "invoice.pdf",            // file name as appears in email
    "application/pdf");      // content type

// add RTF attachment
m.addAttachment(
    "/rtf_doc/product.rtf",    // file to deliver
    "product.rtf",            // file name appears in the email
    "application/rtf");      // content type

// add XML attachment
m.addAttachment(
    "/xml_doc/data.xml",      // file to deliver
    "data.xml",              // file name appears in the email
    "text/xml");            // content type

// If you want to attach HTML documents, use addHtmlAttachment().
// This method automatically resolves the image references
// in your HTML document and attaches those images.
m.addHtmlAttachment("/html_doc/invoice.html");

// add the attachment to the request
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

```
Attachment m = new Attachment();
    m.addHtmlAttachment("/path/to/my.html");
    :
    :
    req.addProperty(DeliveryPropertyDefinitions.SMTP_ATTACHMENT_FIRST,
"true");
    :
```

### Using a String Object as the e-Mail Body

You can use a String object for the e-mail body. This may be useful if you want to include a message with your attached files. The following sample code will deliver the message "Please find the attached invoice." in the e-mail body and one PDF document "invoice.pdf" as an attachment.

```
// 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, "Invoice");
// set SMTP server host
req.addProperty(
    DeliveryPropertyDefinitions.SMTP_HOST, "mysmtpost");
// set the sender email address
req.addProperty(DeliveryPropertyDefinitions.SMTP_FROM,
"myname@mydomain.com");
// set the destination email address
req.addProperty(
    DeliveryPropertyDefinitions.SMTP_TO_RECIPIENTS,
"user1@mydomain.com, user2@mydomain.com");
// set the document to deliver
req.setDocument("Please find the attached invoice. ", "UTF-8");

// create Attachment
Attachment m = new Attachment();
// add attachments
m.addAttachment(
    "/pdf_doc/invoice.pdf",           // file to deliver
    "invoice.pdf",                   // file name appears in the
email
    "application/pdf");             // content type
// add the attachment to the request
req.addProperty(DeliveryPropertyDefinitions.SMTP_ATTACHMENT, m);

// submit the request
req.submit();
// close the request
req.close();

:
:
```

### Using an HTML Document as the e-Mail Body

You can also use an HTML document for the e-mail body. The utility automatically

resolves the local image references in your HTML document and attaches those images also.

Sample usage is as follows:

```
// 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, "Invoice");
// set SMTP server host
req.addProperty(
    DeliveryPropertyDefinitions.SMTP_HOST, "mysmtphost");
// set the sender email address
req.addProperty(DeliveryPropertyDefinitions.SMTP_FROM,
"myname@mydomain.com");
// set the destination email address
req.addProperty(
    DeliveryPropertyDefinitions.SMTP_TO_RECIPIENTS,
"user1@mydomain.com, user2@mydomain.com" );

// set the content type of the email body
req.addProperty(DeliveryPropertyDefinitions.SMTP_CONTENT_TYPE,
"text/html");
// set the document file name appeared in the email
req.addProperty(DeliveryPropertyDefinitions.SMTP_CONTENT_FILENAME,
"body.html");
// set the document to deliver
req.setDocument("/document/invoice.html");

// submit the request
req.submit();
// close the request
req.close();

:
:
```

## Providing Username and Password for Authentication

If the SMTP server requires authentication, you can specify the username and password to the delivery request.

### Example

```
:
req.addProperty(DeliveryPropertyDefinitions.SMTP_USERNAME, "scott");
req.addProperty(DeliveryPropertyDefinitions.SMTP_PASSWORD, "tiger");
:
```

## 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/> for more information about CUPS. See *Setting Up CUPS*, page 7-49 for additional information about setting up CUPS in your system.

To print out your document with the IPP, you need to transform your document into the format that the target IPP printers or servers can understand before the delivery. For example, if the target printer is a Postscript printer, you must transform your document to Postscript format. Usually, printers do not natively understand PDF, RTF, Excel or Word document formats. The Delivery API itself does not provide the document format transformation functionality, but it does offer document filter support for this purpose. See *Document Filter Support*, page 7-35 for more information.

Following is a code sample for delivery to a printer:

#### Example

```
// 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.

---

Property	Description
IPP_HOST	Required Enter the host name.

---

Property	Description
IPP_PORT	Optional Default is 631.
IPP_PRINTER_NAME	Required Enter the name of the printer that is to receive the output. <ul style="list-style-type: none"> <li>• If you use CUPS with the default setup, enter the printer name as <code>/printers/&lt;printer-name&gt;</code></li> <li>• If you use the Microsoft Internet Information Service (IIS) with the default setup, enter the printer name as <code>/printers/&lt;printer-name&gt;/printer</code></li> </ul>
IPP_AUTHTYPE	Optional Valid values for authentication type are: IPP_AUTHTYPE_NONE - no authentication (default) IPP_AUTHTYPE_BASIC - use HTTP basic authentication IPP_AUTHTYPE_DIGEST - use HTTP digest authentication
IPP_USERNAME	Optional Enter the username for HTTP authentication.
IPP_PASSWORD	Optional Enter the password for HTTP authentication.

Property	Description
IPP_ENCTYPE	<p data-bbox="971 306 1065 333">Optional</p> <p data-bbox="971 363 1455 422">The encryption type can be set to either of the following:</p> <p data-bbox="971 447 1382 506">IPP_ENCTYPE_NONE - no encryption (default)</p> <p data-bbox="971 531 1455 558">IPP_ENCTYPE_SSL - use Secure Socket Layer</p>
IPP_USE_FULL_URL	<p data-bbox="971 606 1065 634">Optional</p> <p data-bbox="971 659 1455 751">Set to "true" to send the full URL for the HTTP request header. Valid values are "true" or "false" (default).</p>
IPP_USE_CHUNKED_BODY	<p data-bbox="971 800 1065 827">Optional</p> <p data-bbox="971 852 1455 945">Valid values are "true" (default) to use HTTP chunked transfer coding for the message body, or "false".</p>
IPP_ATTRIBUTE_CHARSET	<p data-bbox="971 993 1065 1020">Optional</p> <p data-bbox="971 1045 1406 1104">Attribute character set of the IPP request. Default is "UTF-8".</p>
IPP_NATURAL_LANGUAGE	<p data-bbox="971 1152 1065 1180">Optional</p> <p data-bbox="971 1205 1390 1264">The natural language of the IPP request. Default is "en".</p>
IPP_JOB_NAME	<p data-bbox="971 1312 1065 1339">Optional</p> <p data-bbox="971 1365 1268 1394">Job name of the IPP request.</p>
IPP_COPIES	<p data-bbox="971 1442 1065 1470">Optional</p> <p data-bbox="971 1495 1463 1554">Define the number of copies to print (example: "1", "5", "10"). Default is 1.</p>

Property	Description
IPP_SIDES	<p data-bbox="873 306 971 333">Optional</p> <p data-bbox="873 363 1365 453">Enable two-sided printing. This setting will be ignored if the target printer does not support two-sided printing. Valid values are:</p> <ul data-bbox="873 478 1365 982" style="list-style-type: none"> <li data-bbox="873 478 1284 506">• IPP_SIDES_ONE_SIDED - default</li> <li data-bbox="873 548 1365 638">• IPP_SIDES_TWO_SIDED_LONG_EDGE - prints both sides of paper for binding long edge.</li> <li data-bbox="873 680 1365 770">• IPP_SIDES_TWO_SIDED_SHORT_EDGE - prints both sides of paper for binding short edge.</li> <li data-bbox="873 812 1365 875">• IPP_SIDES_DUPLEX : Same as IPP_SIDES_TWO_SIDED_LONG_EDGE.</li> <li data-bbox="873 917 1365 980">• IPP_SIDES_TUMBLE : Same as IPP_SIDES_TWO_SIDED_SHORT_EDGE</li> </ul>
IPP_ORIENTATIONS	<p data-bbox="873 1073 971 1100">Optional</p> <p data-bbox="873 1129 1365 1220">Sets the paper orientation. This setting will be ignored if the target printer does not support orientation settings. Valid values are:</p> <p data-bbox="873 1245 1338 1272">IPP_ORIENTATIONS_PORTRAIT (default)</p> <p data-bbox="873 1297 1268 1325">IPP_ORIENTATIONS_LANDSCAPE</p>
IPP_DOCUMENT_FORMAT	<p data-bbox="873 1373 971 1400">Optional</p> <p data-bbox="873 1430 1365 1482">The target printer must support the specified format. Valid values are:</p> <p data-bbox="873 1514 1338 1541">IPP_DOCUMENT_FORMAT_POSTSCRIPT</p> <p data-bbox="873 1566 1338 1593">IPP_DOCUMENT_FORMAT_PLAINTEXT</p> <p data-bbox="873 1619 1240 1646">IPP_DOCUMENT_FORMAT_PDF</p> <p data-bbox="873 1671 1354 1724">IPP_DOCUMENT_FORMAT_OCTETSTREAM (default)</p>

Property	Description
IPP_MEDIA	<p>You can choose either the paper size or the tray number. If you do not specify this option, the default media of the target printer will be used. It will be ignored if the target printer doesn't support the media option. Valid values are:</p> <ul style="list-style-type: none"> <li>• IPP_MEDIA_TRAY1 : Media on tray 1</li> <li>• IPP_MEDIA_TRAY2 : Media on tray 2</li> <li>• IPP_MEDIA_TRAY3 : Media on tray 3</li> <li>• IPP_MEDIA_A3 : A3 Media</li> <li>• IPP_MEDIA_A4 : A4 Media</li> <li>• IPP_MEDIA_A5 : A5 Media</li> <li>• IPP_MEDIA_B4 : B4 Media</li> <li>• IPP_MEDIA_B5 : B5 Media</li> </ul>
IPP_PAGE_RANGES	<p>Specify page ranges to print. By default, all pages are printed. Example valid values are:</p> <ul style="list-style-type: none"> <li>• "3" : prints only page 3.</li> <li>• "2-5" : prints pages 2-5.</li> <li>• "1,3-5" : print page 1 and 3-5.</li> </ul>

### 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"

If you use CUPS with the default setup, the typical property settings are as follows:

- IPP\_HOST : <host-name>
- IPP\_PORT : 631
- IPP\_PRINTER\_NAME : /printers/<printer-name>

If you use the Microsoft Internet Information Service (IIS) with the default setup, the typical property settings are as follows:

- IPP\_HOST : <host-name>
- IPP\_PORT : 80
- IPP\_PRINTER\_NAME : /printers/<printer-name>/.printer

## 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.org/directory/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,
"99999999");
    // 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:

Property	Description
IPP_PHONE_NUMBER	Required Enter the fax number.

## 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.

<b>Property</b>	<b>Description</b>
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.

Property	Description
WEBDAV_ENCTYPE	<p>Optional</p> <p>Valid values for encryption type are:</p> <p>WEBDAV_ENCTYPE_NONE - no encryption (default)</p> <p>WEBDAV_ENCTYPE_SSL - use Secure Socket Layer</p>
WEBDAV_USE_FULL_URL	<p>Optional</p> <p>Set to "true" to send the full URL for the HTTP request header. Valid values are "true" or "false" (default).</p>
WEBDAV_USE_CHUNKED_BODY	<p>Optional</p> <p>Valid values are "true" (default) to use HTTP chunked transfer coding for the message body, or "false".</p>
WEBDAV_URL_CHARACTER_ENCODING	<p>Encoding of the URL. It will be used if you use non-ASCII characters in the URL. Set the Java-supported encoding string for the value.</p>

## Deliver Your Documents Using FTP

The following is sample code for delivery to a FTP server:

## Example

```
// 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, "xdo");
req.addProperty(DeliveryPropertyDefinitions.FTP_PASSWORD, "xdo");
// 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.

Property	Description
FTP_HOST	Required Enter the server host name.
FTP_PORT	Optional Enter the server port number. Default is 21.
FTP_USERNAME	Required Enter the login user name to the FTP server.
FTP_PASSWORD	Required Enter the login password to the FTP server.
FTP_REMOTE_DIRECTORY	Required Enter the directory to which to deliver the document (example: /pub/)

Property	Description
FTP_REMOTE_FILENAME	Required  Enter the document file name for the remote server.
FTP_BINARY_MODE	Optional  Valid values are "true" (default) or "false".

## 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();
```

<b>Property</b>	<b>Description</b>
SFTP_HOST	Required Enter the target server host name.
SFTP_PORT	Optional Enter the target server SSH port number. Default is 22.
SFTP_USERNAME	Required Enter the login user name.
SFTP_PASSWORD	Required if you choose the SFTP_AUTH_TYPE_PASSWORD authentication type. Enter the login password.
SFTP_REMOTE_DIRECTORY	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.
SFTP_REMOTE_FILENAME	Required Enter the document file name on the remote server.
SFTP_AUTH_TYPE	Set either of the following:  SFTP_AUTH_TYPE_PASSWORD (Default) Requires providing password at login.  SFTP_AUTH_TYPE_PUBLIC_KEY - public key authorization type.
SFTP_PRIVATE_KEY_FILE	Enter the client private key file. Required if you choose SFTP_AUTH_TYPE_PUBLIC_KEY.
SFTP_PRIVATE_KEY_PASSWORD	Enter the client private key password. Required if you choose SFTP_AUTH_TYPE_PUBLIC_KEY.

Property	Description
SFTP_FILE_PERMISSION	Enter the permissions to set for the file being created. Default is 0755.

## 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.

```

:
:
// set public key auth type
req.addProperty(DeliveryPropertyDefinitions.SFTP_AUTH_TYPE,
DeliveryPropertyDefinitions.SFTP_AUTH_TYPE_PUBLIC_KEY);
// set username
req.addProperty(DeliveryPropertyDefinitions.SFTP_USERNAME,
"myname");
// set the client's private key file
req.addProperty(DeliveryPropertyDefinitions.SFTP_PRIVATE_KEY_FILE,
"/path/to/the/key");
// set the client's private key password

req.addProperty(DeliveryPropertyDefinitions.SFTP_PRIVATE_KEY_PASSWORD,
"myPrivateKeyPass");
:
:

```

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,
"myname");
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:

```

:
:
// set public key auth type
req.addProperty(DeliveryPropertyDefinitions.SFTP_AUTH_TYPE,
DeliveryPropertyDefinitions.SFTP_AUTH_TYPE_PUBLIC_KEY);
// set username
req.addProperty(DeliveryPropertyDefinitions.SFTP_USERNAME,
"myname");
// set the client's private key file
req.addProperty(DeliveryPropertyDefinitions.SFTP_PRIVATE_KEY_FILE,
"/path/to/the/key");
// set the client's private key password

req.addProperty(DeliveryPropertyDefinitions.SFTP_PRIVATE_KEY_PASSWORD,
"myPrivateKeyPass");
:
:

```

## 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_FILENAME,
"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.

---

<b>Property</b>	<b>Description</b>
HTTP_METHOD	Optional  Sets the HTTP request method. Valid values are:  HTTP_METHOD_POST (Default)  HTTP_METHOD_PUT
HTTP_CONTENT_TYPE	Optional  The document content type (example: "application/pdf").
HTTP_HOST	Required  Enter the server host name.
HTTP_PORT	Optional  Enter the server port number. The default is 80.
HTTP_REMOTE_DIRECTORY	Required  Enter the remote directory name (example: "/home/").
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

---

Property	Description
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).
HTTP_URL_CHARACTER_ENCODING	Encoding of the URL. It will be used if you use non-ASCII characters in the URL. Set the Java-supported encoding string for the value.

## 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,
"as2hsot");

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.

Property	Description
AS2_FROM	Required. Enter the AS2 message sender.
AS2_TO	Required. Enter the AS2 message recipient.
AS2_SUBJECT	Required. Enter the message subject.

Property	Description
AS2_MESSAGE_COMPRESSION	Default value is False. Enter True to compress the message.
AS2_MESSAGE_SIGNATURE	Default value is False. Enter True to sign the message.
AS2_MESSAGE_ENCRYPTION	Default value is False. Enter True to encrypt the message.
AS2_CONTENT_TYPE	<p>Required.</p> <p>Enter the content type of the document. Valid values are:</p> <ul style="list-style-type: none"> <li>• application/EDIFACT</li> <li>• application/xml</li> </ul>
AS2_ENC_ALGO	<p>The AS2 encryption algorithm. Set one of the following:</p> <ul style="list-style-type: none"> <li>• AS2_ENC_ALGO_RC2_40</li> <li>• AS2_ENC_ALGO_RC2_64</li> <li>• AS2_ENC_ALGO_RC2_128</li> <li>• AS2_ENC_ALGO_DES</li> <li>• AS2_ENC_ALGO_DES_EDE3 (Default)</li> <li>• AS2_ENC_ALGO_AES_128</li> <li>• AS2_ENC_ALGO_AES_192</li> <li>• AS2_ENC_ALGO_AES_256</li> </ul>
AS2_DIGEST_ALGO	<p>Enter the AS2 digest algorithm for signing the messages. Set either of the following:</p> <ul style="list-style-type: none"> <li>• AS2_DIGEST_ALGO_MD5 (Default)</li> <li>• AS2_DIGEST_ALGO_SHA1</li> </ul>

<b>Property</b>	<b>Description</b>
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_SENDER_PKCS12_FILE	Location of the sender's PKCS12 (public/private key) file.
AS2_SENDER_PKCS12_PASSWORD	Password for the sender's PKCS12 (public/private key).
AS2_RECEIVER_CERTIFICATES_FILE	Location of the receiver's certificates file.
AS2_DELIVERY_RECEIPT_DIRECTORY	Directory to store the delivery receipts. This directory must be specified if you wish to receive delivery receipts.
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/)

<b>Property</b>	<b>Description</b>
AS2_HTTP_REMOTE_FILENAME	Required. Enter the remote file name.
AS2_HTTP_AUTHTYPE	Enter the HTTP authentication type. Valid values are: <ul style="list-style-type: none"> <li>AS2_HTTP_AUTHTYPE_NONE - no authentication (Default)</li> <li>AS2_HTTP_AUTHTYPE_BASIC - Use HTTP basic authentication.</li> <li>AS2_HTTP_AUTHTYPE_DIGEST - user HTTP digest authentication.</li> </ul>
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: <ul style="list-style-type: none"> <li>AS2_HTTP_ENCTYPE_NONE - no encryption (default)</li> <li>AS2_HTTP_ENCTYPE_SSL - use secure socket layer (SSL)</li> </ul>
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.

Property	Description
AS2_HTTP_PROXY_AUTHTYPE	<ul style="list-style-type: none"> <li>AS2_HTTP_AUTHTYPE_NONE - no authentication (Default)</li> <li>AS2_HTTP_AUTHTYPE_BASIC - Use HTTP basic authentication.</li> <li>AS2_HTTP_AUTHTYPE_DIGEST - user HTTP digest authentication.</li> </ul>
AS2_HTTP_PROXY_USERNAME	Enter the username for proxy authentication.
AS2_HTTP_PROXY_PASSWORD	Enter the password for HTTP proxy authentication.

## Delivery Receipt

The AS2 server always issues an AS2 delivery receipt for each AS2 request. Set the AS2\_DELIVERY\_RECEIPT\_DIRECTORY property to specify the location to store the delivery receipts. If you do not specify this directory, delivery receipts will be ignored. Example code for setting the delivery receipt directory is as follows:

```

:
:
// Set the delivery receipt directory

req.addProperty(DeliveryPropertyDefinitions.AS2_DELIVERY_RECEIPT_DIRECTORY, "/my/receipt/dir");
:
:

```

## 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 e-mail address where the delivery receipt will be delivered after processing. You must set up the HTTP server or e-mail address to receive the delivery receipts.

The Delivery API can track down the asynchronous request if you specify the e-mail address for the AS2\_ASYNC\_ADDRESS. If you provide the e-mail account information to the Delivery API, the Delivery API will periodically check the e-mail account to obtain the delivery receipt. Sample code for this is as follows:

## Example

```
:
:
// Set the email address - async request
req.addProperty(DeliveryPropertyDefinitions.AS2_ASYNC_ADDRESS,
"async_target@acme.com");

// Set the delivery receipt directory

req.addProperty(DeliveryPropertyDefinitions.AS2_DELIVERY_RECEIPT_DIRECTO
RY, "/my/receipt/dir");

// Set the email server information where the delivery receipt will be
delivered to.
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.addProperty(
    DeliveryPropertyDefinitions.AS2_ASYNC_EMAIL_SERVER_FOLDER_NAME,
"inbox");

// set the document
req.setDocument("/document/myEDIdoc");

// submit the request with the DeliveryResponseListener
req.submit(myDeliveryListener);
:
:
```

Note that as shown in the preceding code, you must use the Delivery API's asynchronous delivery request mechanism to track down the asynchronous requests. See *Asynchronous Delivery Requests*, page 7-34 for more information.

## Document Signing

The Delivery API allows you to sign a document for the secure transaction. This is based on the public key architecture, so you must set up the following:

- Sender side: sender's public/private keys  
Sender must have sender's public/private keys in a PKCS12 standard file. The file extension is .p12. Place that file in your local system where you want to run the Delivery API.
- Receiver side (AS2 server side) : sender's public key certificate  
The receiver must have the sender's public key certificate. Installing certificates on the AS2 server can vary depending on your server. Generally, you must copy the .der or .cer certificates to a particular location. Consult your AS2 server manual for the procedure.

Once you have completed the setup, you can sign your document by setting properties in the delivery request. Sample code for this is as follows:

```
:
:
// Signing the document
req.addProperty(DeliveryPropertyDefinitions.AS2_MESSAGE_SIGNATURE,
"true");
req.addProperty(DeliveryPropertyDefinitions.AS2_SENDER_PKCS12_FILE,
"/path/to/mykey.p12");
req.addProperty(DeliveryPropertyDefinitions.AS2_SENDER_PKCS12_PASSWORD,
"welcome");
:
:
```

## Document Encryption

The Delivery API allows you to encrypt documents for the secure transaction. This is based on the public key architecture, so you need to set up the following:

- Sender's side: Receiver's public key certificate  
The sender side must have the receiver's public key certificate file. The file extension is .der or .cer. Place that file in your local system where you want to run the Delivery API. Please consult the manual of your AS2 server for the procedure to obtain the AS2 server's public key certificate.
- Receiver's side (AS2 server side): Receiver's public/private keys  
The receiver side (AS2 Server) must have the receiver's public/private keys. Please consult the manual of your AS2 server for the procedure to set up keys.

Once set up, you can encrypt your document by setting properties in the delivery request. The sample code is as follows:

```
:
:
// Encrypting the document
req.addProperty(DeliveryPropertyDefinitions.AS2_MESSAGE_ENCRYPTION,
"true");

req.addProperty(DeliveryPropertyDefinitions.AS2_RECEIVER_CERTIFICATES_FILE,
"/path/to/server-certificate.der");
:
:
```

## Delivering Documents Using an External Command

The Delivery API supports the use of external, Operating System (OS) native commands to deliver documents.

Specify your OS native command with the `{file}` placeholder. At runtime, this placeholder will be replaced with the document file name.

The delivery status is determined by the exit value of the OS command. If the value is

'0', the request is marked successful.

Sample code is as follows:

```
// create delivery manager instance
    DeliveryManager dm = new DeliveryManager();
    // create a delivery request
    DeliveryRequest req =
dm.createRequest(DeliveryManager.TYPE_EXTERNAL);
    // set the OS native command for delivery

req.addProperty(ExternalDeliveryPropertyDefinitions.EXTERNAL_DELIVERY_CO
MMAND,
    "/usr/bin/lp -d myprinter {file}");
    // set the document
    req.setDocument("/document/test.pdf");

    // submit the request
    req.submit();
    // close the request
    req.close();
```

The following property is supported and defined in DeliveryPropertyDefinitions:

Property	Description
EXTERNAL_DELIVERY_COMMAND	Required. Enter the OS native command for delivery.

## Delivering Documents to the Local File System

The Delivery API supports the delivery of documents to the local file system where the Delivery API runs. The command copies the file to the location you specify.

The following sample code copies the file /document/test.pdf to /destination/document.pdf.

```
// create delivery manager instance
    DeliveryManager dm = new DeliveryManager();
    // create a delivery request
    DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_LOCAL);
    // set the document destination in the local filesystem.

req.addProperty(ExternalDeliveryPropertyDefinitions.LOCAL_DESTINATION,
"/destination/document.pdf");
    // set the document to deliver.
    req.setDocument("/document/test.pdf");

    // submit the request
    req.submit();
    // close the request
    req.close();
```

The following property is supported and defined in DeliveryPropertyDefinitions:

Property	Description
LOCAL_DESTINATION	Required. Full path to the destination file name in the local file system.

## 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 7-45.

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();
```

## Asynchronous Delivery Requests

The Delivery API provides the ability to run the delivery requests asynchronously by registering the callback functions.

You can create your own callback logic by implementing the `DeliveryResponseListener` interface. You must implement the `responseReceived()` method. You can implement your logic in this method so that it will be called when the delivery request is finished.

Sample code is as follows:

```
import oracle.apps.xdo.delivery.DeliveryResponseListener;

class MyListener implements DeliveryResponseListener
{
    public void responseReceived(DeliveryResponse pResponse)
    {
        // Show the status to the System.out
        System.out.println("Request done!");
        System.out.println("Request status id : " +
pResponse.getStatus());
        System.out.println("Request status msg : " +
pResponse.getStatusMessage());
    }
}
```

Once you implement the callback, you can pass your callback when you call the `submit()` method of your `DeliveryRequest`. If you call the `submit()` with the callback, the delivery process will start in the background and the `submit()` method will immediately return the control. Sample code follows:

```

// create delivery manager instance
DeliveryManager dm = new DeliveryManager();

// create a delivery request
DeliveryRequest req =
dm.createRequest(DeliveryManager.TYPE_IPP_PRINTER);
:
:
// submit request with the callback logic
req.submit(new MyListener());
:
:

```

## Document Filter Support

The Delivery API supports the document filter functionality for all the supported protocols. This functionality allows you to call the native OS command to transform the document before each delivery request. To specify the filter, pass the native OS command string with the two placeholders for the input and output filename: {infile} and {outfile}. You can set your filter in your delivery request as a delivery property. Following are two samples:

```

// The easiest filter, just copy the file :)
req.addProperty(DeliveryPropertyDefinitions.FILTER, "cp {infile}
{outfile}");

```

```

// Call "pdftops" utility to transform the PDF document into Postscript
format
req.addProperty(DeliveryPropertyDefinitions.FILTER, "pdftops {infile}
{outfile}");

```

Alternatively, you can also specify the filter for each server in the configuration file (see Configuration File Support, page 7-45). In this case, the server will always use this filter for the requests to this server:

```

:
:
<server name="printer1" type="ipp_printer" default="true">
<uri>ipp://myserver:80/printers/MyPrinter1/.printer</uri>
<filter>pdftops {infile} {outfile}</filter>
</server>
:
:

```

This is useful especially if you are trying to call IPP printers directly or IPP printers on Microsoft Internet Information Service (IIS) because those printers usually do not accept PDF documents, but only limited document formats. With this functionality, you can call any of the native OS commands to transform the document to the format that the target printer can understand. For example, if you need to call the HP LaserJet printer setup on the Microsoft IIS from Linux, you can set Ghostscript as a filter to transform the PDF document into the format that the HP LaserJet can understand.

```
// specify filter
req.addProperty(DeliveryPropertyDefinitions.FILTER,
"gs -q -dNOPAUSE -dBATCH -sDEVICE=laserjet -sOutputFile={outfile}
{infile}");
```

Note that to use this functionality you must set the buffering mode must be enabled and a temporary directory must be specified. See Configuration File Support, page 7-45 for information on setting these properties.

## Date Expression Support

Three properties support date expressions. Use the date expression if you want to name a file by the date, and have the date automatically set at runtime.

The properties that support date expressions are:

- SMTP\_CONTENT\_FILENAME
- FTP\_REMOTE\_FILENAME
- WEBDAV\_REMOTE\_FILENAME

The supported date expressions are:

- %y : 4 digit year (ex, 1972, 2005)
- %m : 2 digit month (00 - 12)
- %d : 2 digit date (00 - 31)
- %H : 24h based 2 digit hour (00 - 24)
- %M : 2 digit minute (00 - 59)
- %S : 2 digit sec (00 - 59)
- %l : 3 digit millisecc (000 - 999)

For example, if you specify `my_file_%y%m%d.txt` for the filename, the actual filename will be `my_file_20051108.txt` for November 8, 2005. All undefined expressions will be translated into 0 length string, for example, if you specify `my_file_%a%b%c.txt`, it would generate `my_file_.txt`. You can escape the '%' character by passing '%%'.

## Internationalization

The Delivery Server API supports following internationalization features for the listed delivery channels:

### **SMTP**

- Specify character encoding for the main document with `SMTP_CONTENT_TYPE`.
- Specify character encoding for the attachments by passing content type when you call `addAttachment()` method.
- Specify the character encoding for email To/From/CC/Subject with `SMTP_CHARACTER_ENCODING` property. The default value is "UTF-8".

### **IPP**

- Specify character encoding for the IPP attributes by using `IPP_ATTRIBUTE_CHARSET` property. The default value is "UTF-8".
- Specify `IPP_URL_CHARACTER_ENCODING` property for encoding non-ASCII letters in a URL.

### **WebDAV**

- Specify `WEBDAV_URL_CHARACTER_ENCODING` property for encoding non-ASCII letters in a URL.

### **FTP**

- The FTP delivery channel automatically detects the internationalization support in the target FTP server. You can specify a non-ASCII directory name and file name only if the FTP server supports internationalization (see RFC 2640 for more detail). In that case, the UTF-8 encoding will be used automatically. If the server does not support internationalization and you specify a non-ASCII value, an exception will be thrown during the delivery process.

### **HTTP**

- You can specify `WEBDAV_URL_CHARACTER_ENCODING` property for encoding non-ASCII letters in a URL.

## **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

```
// create delivery manager instance
    DeliveryManager dm = new DeliveryManager();

    // create a delivery request
    DeliveryRequest req =
dm.createRequest(DeliveryManager.TYPE_IPP_PRINTER);
    :
    :
    // 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:

Property	Description
BUFFERING_MODE	Valid values are "true" (default) and "false". See Direct and Buffering Modes, page 7-33 for more information.
TEMP_DIR	Define the location of the temporary directory.
CA_CERT_FILE	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.

### 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_SMTPEMAIL);
:
:
```

## 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

```
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:

```
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

```
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));

            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_IO_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 7-45 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</ch
annel>
```

```
</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

```
// 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 sever 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.

## Supported Configuration File Properties and Elements

The following properties are supported in the `<properties>` section:

- `ds-temp-dir`: temporary directory location.
- `ds-buffering`: specify true or false for buffering mode.
- `ds-ca-cert-file`: specify the SSL certification file location.

The following elements are supported for `<server type="ipp_printer">` and `<server type="ipp_fax">`

- `<host>`
- `<port>`
- `<printerName>`
- `<uri>`
- `<username>`

- <password>
- <authType>
- <encType>
- <proxyHost>
- <proxyPort>
- <proxyUsername>
- <proxyPassword>
- <proxyAuthType>
- <filter>

The following elements are supported for <server type="smtp\_email">

- <host>
- <port>
- <uri>
- <username>
- <password>
- <authType>
- <filter>

The following elements are supported for <server type="webdav">

- <host>
- <port>
- <uri>
- <username>
- <password>
- <authType>
- <encType>

- <proxyHost>
- <proxyPort>
- <proxyUsername>
- <proxyPassword>
- <proxyAuthType>
- <filter>

The following elements are supported for <server type="ftp"> and <server type="sftp">

- <host>
- <port>
- <uri>
- <username>
- <password>
- <filter>

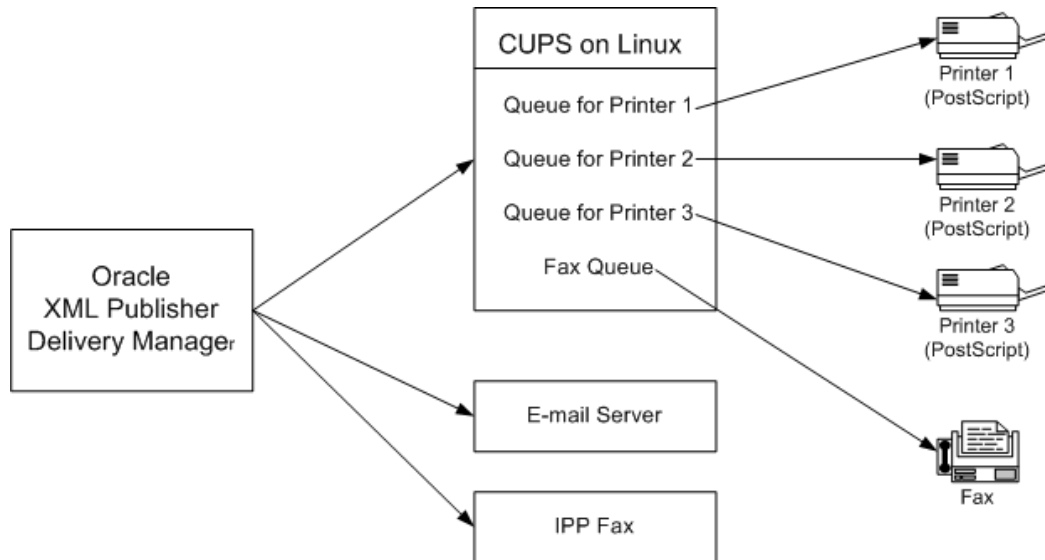
The following elements are supported for <server type="external">

- <command>
- <filter>

## Setting Up CUPS

The delivery manager requires Common UNIX Printing System (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 7-53.
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, 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 want 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
p_DataSource	<p>(Required) The following XML data source types are supported:</p> <ul style="list-style-type: none"> <li>DATA_SOURCE_TYPE_REQUEST_ID: concurrent program request ID</li> <li>DATA_SOURCE_TYPE_FILE - XML data file</li> <li>DATA_SOURCE_TYPE_BLOB - BlobDomain</li> <li>DATA_SOURCE_TYPE_DOCUMENT - Final document for preview</li> </ul> <p>These types are defined in the xdo.oa.common.DocumentHelper class.</p> <p>Additional parameters may be required depending on the type of data source. These parameters are described in the next section.</p>
p_TemplateCode	<p>(Optional) If set to null, the UI will provide a list from which to select 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</p>
p_TemplateAppsShortName	<p>Required if p_TemplateCode is not null. Enter the Application short name of the application to which the template is assigned in the Template Manager. Example: AR</p>
p_Locale	<p>(Optional) If null, the UI will provide a list to select available locales for the selected template. The value "Default" can be entered to select the default template locale.</p>
p_OutputType	<p>(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.</p>
p_XDORegionHeight	<p>Height of the XDO common region window expressed as a percentage. Example: 60%</p>

## 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.

Parameter	Description
p_RequestId	Enter the concurrent request ID.

The following parameters are required when the parameter p\_DataSource is DocumentHelper.DATA\_SOURCE\_TYPE\_FILE.

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
p_AbsolutePath	Enter the absolute path for the XML data file.

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.

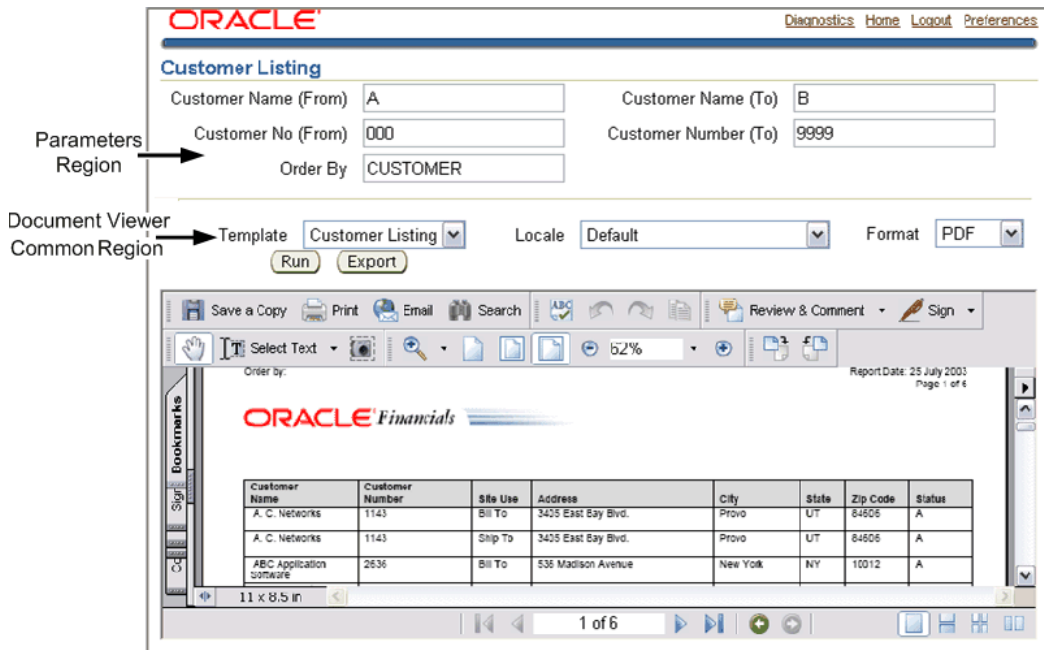
Parameter	Description
p_AbsolutePath	Enter the absolute path for the document file.
p_DocumentType	Enter the document type to determine the correct content type. Valid values are: PDF, RTE, EXCEL, HTML.

## 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:

```

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);

```

## DocumentHelper.exportDocument

This method can be called from any event, such as the submit button to export the document.

```

public static void exportDocument(

    OAPageContext pageContext,
    String appShortName,
    String templateCode,
    String language,
    String territory,
    InputStream inputStream,
    String outputType,
    Properties properties)

```



---

# Setting Up XML Publisher

## Overview

The following table shows the required and optional setup steps for XML Publisher. You may have already completed some of these steps during install or upgrade.

Task	Required/Optional
Assign the XML Publisher responsibility to users requiring Administration privileges	Required
Specify a temporary directory	Required
Install XML Publisher desktop components	Recommended for report designers
Enable PDF printing in Oracle Applications	Required

### Assign the XML Publisher Responsibility to Users

Assign the XML Publisher Administration responsibility (key=XDO\_ADMINISTRATION) to users who must access the Template Manager and Administration pages.

For information on assigning a responsibility to a user, see the *Oracle Applications System Administrator's Guide - Security*.

### Specify a Temporary Directory

Use the Administration interface to assign a temporary directory for the site level. Navigate from the XML Publisher Administrator responsibility by selecting Administration, then Configuration, then General Properties.

If you are running XML Publisher on multiple middle tier machines, then you set the

temporary directory at the server level for each machine. To set the temporary directory at the server level, you must specify it in a configuration file. For information on setting up a configuration file for XML Publisher, see *Setting Up the Configuration File*, page B-1.

### **Install XML Publisher Desktop Components**

XML Publisher supplies two desktop tools to increase your productivity. The Oracle Business Intelligence Publisher Template Builder for Word Add-in is an extension to Microsoft Word that simplifies the development of RTF templates. It automates many of the template building procedures.

The Template Viewer provides advanced preview capabilities to enable you to view your template as a completed report as you build it.

The XML Publisher desktop components are available from a separate patch on Oracle *MetaLink*. See *About Oracle XML Publisher Release 5.6.3, Note: 422508.1* for the patch number. These tools are recommended for anyone designing RTF templates.

### **Enable PDF Printing in Oracle Applications**

The "PDF Publisher" print style and PASTA\_PDF printer driver provide the capability to print PDF files using a 3rd party utility. You can use this style and driver to print a generated PDF. "--Pasta Universal Printer" type has been associated with the style and driver for ease of use. See the *Pasta User's Guide 3.0* on Oracle *MetaLink* for more information.

---

## XML Publisher Configuration File

### XML Publisher Configuration File

Properties set in the Administration interface (see Administration, page 4-1) 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 -->
<font>
  <!-- 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>
</font>
</config>
```

## How to Read the Element Specifications

The following is an example of an element specification:

```
<Element Name Attribute1="value"
  Attribute2="value"
  AttributeN="value"
  <Subelement Name1/>[occurrence-spec]
  <Subelement Name2>...</Subelement Name2>
  <Subelement NameN>...</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/">
  <font> ... </font> [0..n]
  <property> ... </property> [0..n]
</config>
```

## Attributes

<b>version</b>	The version number of the configuration file format. Specify 1.0.0.
<b>xmlns</b>	The namespace for XML Publisher's configuration file. Must be <code>http://xmlns.oracle.com/oxp/config/</code>

## Description

The root element of the configuration file. The configuration file consists of two parts:

- Properties (`<property>` elements)
- Font definitions (`<font>` elements)

The `<font>` and `<property>` 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 internal property name (key) to the name attribute and the value to the element value. See List of Properties, page 4-2 for the list of the internal property names.

### Example

```
<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 4-2 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:

Property Name	Internal Name	Default Value	Description
Compress PDF output	pdf-compression	True	Specify "True" or "False" to control compression of the output PDF file.

To specify this property in the configuration file, enter the following:

```
<properties>
  <property name="pdf-compression">>false</property>
</properties>
```

## Properties Defined Only in the Configuration File

The following properties are not available in the Administration interface and must be set using the configuration file.

### PDF Form Processing Engine Properties

Property Name	Default Value	Description
<code>remove-pdf-fields</code>	false	Specify "true" to remove PDF fields from the output. When PDF fields are removed, data entered in the fields cannot be extracted. For more information, see Setting Fields as Updateable or Read Only, <i>Oracle XML Publisher Report Designer's Guide</i> .
<code>all-field-readonly</code>	true	By default, XML Publisher sets all fields in the output PDF of a PDF template to be read only. If you want to set all fields to be updateable, set this property to "false". For more information, see Setting Fields as Updateable or Read Only, <i>Oracle XML Publisher Report Designer's Guide</i> .
<code>all-fields-readonly-asis</code>	false	Set this property to "true" if you want to maintain the "Read Only" setting of each field as defined in the PDF template. This property overrides the settings of <code>all-field-readonly</code> . For more information, see Setting Fields as Updateable or Read Only, <i>Oracle XML Publisher Report Designer's Guide</i> .

### PDF Document Merger Property

Property Name	Default Value	Description
<code>pdf-tempfile-max-size</code>	unlimited	This property sets the maximum size for the temporary file used during batch processing by the PDF Document Merger. Enter the maximum size in bytes. For more information, see PDF Document Merger, page 6-31.

## FO Processor Property

Property Name	Default Value	Description
<code>rtf-adj-table-border-overlap</code>	<code>false</code>	Sets the behavior of the top/bottom borders of adjacent tables. If you want the top and bottom borders of adjacent tables to overlap (for example, if you want to repeat a table multiple times and make the whole repeating area appear to be a single table), set this property to "true."

## Font Definitions

Font definitions include the following elements:

- `<font>`
- `<font-substitute>`
- `<truetype>`
- `<type1>`

For the list of Truetype and Type1 fonts, see *Predefined Fonts*, page 4-17.

### `<font>` element

The `<font>` element is structured as follows:

```
<font locales="cdata">  
  <font> ... </font> [0..n]  
  <font-substitute> ... </font-substitute> [0..n]  
</font>
```

### Attributes

**locales** Specify the locales for this font definition. This attribute is optional.

### Description

The `<font>` element defines a set of fonts. Specify the `locales` attribute to define locale-specific fonts.

### Example

```
<!-- Font definitions for all locales -->
<font>
  ..Font definitions here...
</font>

<!-- Korean-specific font definitions -->
<font locales="ko-KR">
  ... Korean Font definitions here...
</font>
```

## <font> element

Following is the structure of the <font> element:

```
<font family="cdata" style="normalitalic"
weight="normalbold">
  <truetype>...</truetype>
or <type1> ... <type1>
</font>
```

### Attributes

<b>family</b>	Specify any family name for the font. If you specify "Default" for this attribute, you can define a default fallback font. The <b>family</b> attribute is case-insensitive.
<b>style</b>	Specify "normal" or "italic" for the font style.
<b>weight</b>	Specify "normal" or "bold" for the font weight.

### Description

Defines a 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

```
<!-- 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

<b>name</b>	Specify the name of the font to be substituted.
-------------	-------------------------------------------------

## Description

Defines a font substitution. This element is used to define fonts for the PDF Form Processor.

### Example

```
<font-substitute name="MSGothic">
  <truetype path="/fonts/msgothic.ttc" ttccno=0"/>
</font-substitute>
```

## <type1> element

The form of the <type1> element is as follows:

```
<type1 name="cdata"/>
```

## Attributes

<b>name</b>	Specify one of the Adobe standard Latin1 fonts, such as "Courier".
-------------	--------------------------------------------------------------------

## Description

<type1> element defines an Adobe Type1 font.

### Example

```
<!--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:

```
- xdo.cfg -
<fonts locales="zh"> // section for all zh-* locale
  <font family="xxx" style="normal" weight="normal">
    <truetype path="xxx4zh.ttf" />
  </font>
  <font family="yyy" style="normal" weight="normal">
    <truetype path="yyy4zh.ttf" />
  </font>
</font>
<fonts locales="zh-TW"> // override "zh" section for "zh-TW"
  // must be after "zh" section
  <font family="yyy" style="normal" weight="normal">
    <truetype path="yyy4zh-TW.ttf" />
  </font>
</font>
```

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>
targetldtfile.ldt XMLLP ATTRIBUTES
```

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
XMLPData.ldt XDO_DS_DEFINITIONS APPLICATION_SHORT_NAME=AR
```

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/patch/115/import/xdotmpl.lct XMLPData.ldt 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/pword@mydb 0 Y DOWNLOAD
$XDO_TOP/patch/115/import/xdotmpl.lct XMLPData.ldt XDO_DS_DEFINITIONS
APPLICATION_SHORT_NAME=AR DATA_SOURCE_CODE=ARXCOBLX
TMPL_APP_SHORT_NAME=AR
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.oa.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:

<b>Parameter Name</b>	<b>Description</b>
DOWNLOAD	(Mandatory) The first parameter: DOWNLOAD will be implemented in the feature.
DB_USERNAME	(Mandatory) Database user name (example: apps).
DB_PASSWORD	(Mandatory) Database user password (example: manager).
JDBC_CONNECTION	(Mandatory) JDBC database connection string (example: ap000sun:1521:db222).
LOB_TYPE	(Mandatory) XDO LOB type. Valid values are: TEMPLATE XML_SCHEMA XML_SAMPLE
APPS_SHORT_NAME	(Mandatory) Application short name (example: AR).
LOB_CODE	(Optional) XDO LOB code. Enter either the Template Code or the Data Definition Code (see below).
LCT_FILE	(Optional) This is the control file for XML Publisher metadata (see below).
LANGUAGE	(Mandatory for template files only) ISO two-letter language code (example: en)
TERRITORY	(Mandatory for template files only) ISO two-letter territory code (example: US)
LOG_FILE	(Optional) Enter a file name for the uoutput log file (default: xdotmpl.log).

Parameter Name	Description
DEBUG	(Optional) Turns debug on or off. Valid values are:  true  false (default)

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 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 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 <drv file> \
  -LOG_FILE <log file>
```

Parameter Name	Description
DOWNLOAD	(Mandatory) The first parameter: DOWNLOAD will be implemented in the feature.
DB_USERNAME	(Mandatory) Database user name (example: apps).
DB_PASSWORD	(Mandatory) Database user password (example: manager).
JDBC_CONNECTION	(Mandatory) JDBC database connection string (example: ap000sun:1521:db222).
APPS_SHORT_NAME	(Mandatory) Application short name (example: AR).
LCT_FILE	(Mandatory) Full path to the xdotmpl.lct
DS_CODE	(Optional) Data source code.
LDT_FILE	(Optional) Output LDT file name (default: xdotmpl.ldt)
DRVX_FILE	(Optional) Output DRVX file name (default: xdotmpl.drvx)
LOG_FILE	(Optional) Enter a file name for the output log file (default: xdotmpl.log).
DEBUG	(Optional) Turns debug on or off. Valid values are:  true  false (default)

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 <xdo_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:

Parameter Name	Description
UPLOAD	(Mandatory) The first parameter: UPLOAD will be implemented in the feature.
DB_USERNAME	(Mandatory) Database user name (example: apps).
DB_PASSWORD	(Mandatory) Database user password (example: manager).
JDBC_CONNECTION	(Mandatory) JDBC database connection string (example: ap000sun:1521:db222).

Parameter Name	Description
LOB_TYPE	(Mandatory) XDO LOB type. Valid values are: TEMPLATE XML_SCHEMA XML_SAMPLE
APPS_SHORT_NAME	(Mandatory) Application short name (example: AR).
LOB_CODE	(Mandatory) XDO LOB code. Enter either the Template Code or the Data Definition Code.
NLS_LANG	(Mandatory) Enter the NLS_LANG environment variable.
LANGUAGE	(Optional) ISO two-letter language code (example: en). If NLS_LANGUAGE='TRADITIONAL CHINESE', then cn_TW and if NLS_LANGUAGE='SIMPLIFIED CHINESE' then cn_CN for combination of language and territory.
TERRITORY	(Mandatory) ISO two-letter territory code (example: US), default is '00'.
XDO_FILE_TYPE	(Mandatory) Enter the XDO file type, valid values are: PDF, RTF, XLS, XSL-FO, XSL-HTML, XSL-XML, XSL-TEXT, XSD, XML, RTF-ETEXT
FILE_CONTENT_TYPE	(Optional) Content type of the file (example: text/html, application/pdf)
FILE_NAME	(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/115app/xdo/115/patch/115/publisher/templates )
OWNER	(Optional) Owner of the template. Default is "ORACLE".
CUSTOM_MODE	(Optional) Whether to force update. Valid values are FORCE and NOFORCE (default).
LOG_FILE	(Optional) Enter a file name for the output log file (default: xdotmpl.log).

Parameter Name	Description
DEBUG	(Optional) Turns debug on or off. Valid values are:  true  false (default)
USE_APPS_CONTEXT	(Optional) Whether to use AppsContext or not. Valid values are:  true  false (default)  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.



---

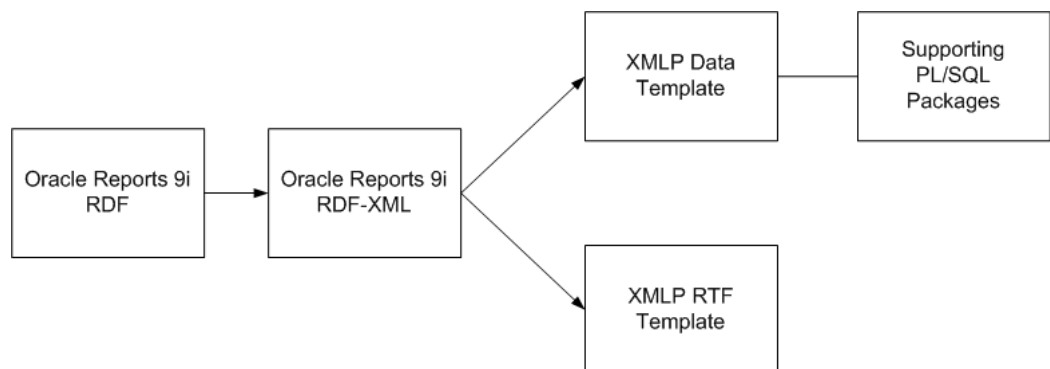
# Oracle Report to XML Publisher Report Migration

## Overview

XML Publisher provides a utility to facilitate the conversion of Oracle Reports to XML Publisher Reports.

In an Oracle Report the data model (data extraction logic) and layout (presentation) are embedded together in a single RDF file. In XML Publisher, the data model and the layout are separate. The migration is therefore a two-step process to convert the RDF file into the separate objects necessary for the XML Publisher report.

The two steps can be combined into a single shell script to convert an Oracle Report; and, the script can be modified to convert multiple Oracle Reports in a given directory. The following figure shows the conversion flow:



## Prerequisites

The migration utility accepts an Oracle Report in XML format. This format is only supported in Oracle Reports 9i and later.

To convert the Oracle Report RDF file to XML format, use either the Oracle Report

Designer or the Oracle Report `rwconverter.exe` utility under `$ORACLE_HOME/bin`.

### Example

The following sample executes the `rwconverter` utility, taking the source report, `raxinv.rdf` and converting it to an RDF-XML format that can be consumed by the XML Publisher conversion utility. Note the `dtype` must be specified as "xmlfile".

```
D:\Oracle_home\BIN>rwconverter batch=yes source= h:\reports\raxinv.rdf
dest= h:\reports\raxinv.xml dtype=xmlfile overwrite=yes
```

## Conversion Process

### Data Model Migration

Use the `DataTemplateGenerator` API to migrate the Oracle Reports Data Model to a Data Template and associated PL/SQL logic to PL/SQL Package (specification and body).

The API can be called through the command line or through a shell script. This will generate following output files:

- Data Template (REPORT.xml)
- Default PL/SQL package specification (REPORTS.pls)
- Default PL/SQL package body (REPORTB.pls)

### Example

```
javaw.exe oracle.apps.xdo.rdfparser.DataTemplateGenerator
H:\report\raxinv.xml
```

Output Files:

- PL/SQL Package: H:\report\raxinvS.pls
- PL/SQL Body: H:\report\raxinvB.pls
- DataTemplate: H:\report\raxinv\_template.xml

### Layout Migration

Use the `RTFTemplateGenerator` API to migrate the Oracle Reports layout to an XML Publisher RTF template.

Because there is no support of PL/SQL in an RTF Template, the process does not migrate any format trigger logic present in the report. Instead, the generator writes all the format trigger code to the log file. You then must implement any corresponding PL/SQL logic as XSL code.

Many Oracle Reports use simple "if" formatting logic that is not difficult to convert. To aid in this process, the resulting RTF template will contain Microsoft Word form fields that hold the format trigger names that are called. These fields will be highlighted in red. You can then refer to the log to find the actual PL/SQL code used in the original

Oracle Report.

The API can be called through the command line or through a shell script. This will generate following output files:

- RTF Template
- Log file

**Example**

```
javaw.exe oracle.apps.xdo.rdfparser.RTFTemplateGenerator  
H:\report\raxinv.xml
```

Output Files:

- RTF Template: H:\ report\ raxinv.rtf
- Log File : H:\report\raxinv.log

## Known Issues

The following are known issues with the conversion APIs:

- Some complex Oracle Reports may result in minor errors in the generated Data Template or PL/SQL that require manual correction.
- Format triggers are not supported. The format trigger logic must be implemented separately though XSLT.
- The Data Template cannot support the following scenario: A formula column references the summary column as a parameter and the summary column belongs to the same Data Source/Data Query. This is not supported because the data template moves all the formula columns to the select statement, therefore the summary column value is not available while executing the formula.

## Batch Conversion

The three components to the migration can be combined into a single shell script to completely automate the process. Furthermore, rather than just have the script run on a single report file, the script can be modified to run on all the reports in a given directory.

This script converts all RDFs in a directory. You need the following Java libraries:

- Collections.zip – available from Sun Microsystems
- xmlparserv2-904 – available from the JAVA\_TOP directories
- A pointer to the Oracle Applications JAVA\_TOP (where the required XML Publisher libraries reside)

The sample annotated script follows:

```
#!/bin/sh
# This script will generate a report for each template in the current
directory
# Create a variable to hold the classpath
classpath="DIR/collections.zip:DIR/xmlparserv2-904.zip:JAVA_TOP
directory"
if [ $# -eq 0 ]
then
    for file in *.rdf
    do
echo "Processing ... $file"
        if test -f $file
        then
# Convert the rdf to xml
            echo yes | $ORACLE_HOME/bin/rwconverter.sh batch=yes
source=$file dest=$file dtype=xmlfile overwrite=yes; \
#Create a variable to hold the new xml file name, this is just a simple
replace
# statement

            xfile="{file//rdf/xml}";
# Generate the data template plus plsqli
            echo yes | /local/java/jdk1.5.0_06/bin/java -classpath
$classpath oracle.apps.xdo.rdfparser.DataTemplateGenerator $xfile;
# Generate the RTF template
            echo yes | /local/java/jdk1.5.0_06/bin/java -classpath
$classpath oracle.apps.xdo.rdfparser.RTFTemplateGenerator $xfile;

        fi
    done

else
    echo usage: $0
    echo this script will generate a data template and supporting plsqli
and an RTF template in the current directory
fi
```

---

# Index

## A

---

administration  
    configuration tab, 4-2  
APIs, 6-1

## B

---

barcode formatting  
    APIs, 6-70  
buffering mode  
    delivery server, 7-33  
bursting engine, 6-44

## C

---

common region, 8-1  
    APIs, 8-6  
concurrent manager, 3-1, 3-1  
    calling the data template, 5-27  
concurrent program  
    for data template, 5-27  
configuration  
    data definition level, 2-5  
    template level, 2-12  
configuration file, B-1  
    <properties> element, B-3  
    <root> element, B-3  
    delivery manager, 7-45  
    structure, B-3  
configuration properties  
    precedence of levels, 4-2  
configuration tab  
    administration, 4-2

Copy Template page, 2-8  
Create Data Definition page, 2-2  
    field definitions, 2-2  
Create Template page  
    field definitions, 2-6  
CUPS setup, 7-49  
currencies  
    defining format sets, 4-21

## D

---

Data Definition  
    creating in Template Manager, 2-2  
data definition properties  
    setting, 2-5  
data template  
    calling, 5-27, 5-29  
    constructing, 5-7  
    setting up concurrent program for, 5-27  
    using the concurrent manager to call, 5-27  
data template definition, 5-3  
default template, 2-7  
default template file, 2-5  
delivery  
    using OS command, 7-31  
delivery channels  
    adding custom, 7-39  
delivery manager  
    configuration file, 7-45  
delivery server, 7-36  
    buffering mode, 7-33  
    date expression, 7-36  
    direct mode, 7-33

- document filter support, 7-35
- global properties, 7-38
- local file system delivery, 7-32

delivery status, 7-37

direct mode

- delivery server, 7-33

document viewer, 8-1

- APIs, 8-6
- implementing, 8-4
- parameters, 8-1

downloading objects, C-1

---

## E

editing templates from the Template Manager, 2-10

e-mail delivery, 7-2

enabling a translation, 2-17

---

## F

fax delivery, 7-14

font definitions

- configuration file, B-6

font fallback

- configuration file, B-8

font fallback mechanism, 4-16

font files

- administration, 4-13
- location, 4-17
- uploading, 4-13

font mapping

- administration, 4-13

FTP delivery, 7-17

---

## G

generating output, 3-1

global properties

- delivery server, 7-38

---

## H

HTTP

- delivering documents over, 7-22

---

## I

incomplete translations, 2-16

---

## J

javadocs

- obtaining, 6-1

---

## L

languages

- adding templates for , 2-10

locales

- configuration file, 4-15

---

## M

merging PDF files, 6-28

---

## O

Oracle reports

- migration to XML Publisher, D-1

Out of memory error

- avoiding, 4-3

---

## P

PDF files

- merging, 6-28

PDF template

- template mapping, 2-5

PDF template mapping, 2-7, 2-11

predefined fonts, 4-17

previewing a template, 2-10

printers

- setup
- Unix/Linux, 7-49

printing, 7-8

progress indicator

- translations, 2-16

properties element

- configuration file, B-3

publishing, 3-1

---

## R

report migration, D-1

runtime properties

- data definition, 2-5

---

## S

- secure ftp
  - delivery, 7-19
- seeded templates
  - deleting, 2-10
  - modifying, 2-9
- setRowsetTag method, 5-31
- setRowsTag method, 5-31
- setSQL method, 5-31
- Standard Request Submission, 3-1
- status indicator
  - translations, 2-16

## **T**

---

- table borders
  - configure overlapping borders, B-6
- template
  - defining in Template Manager, 2-5
- Template Manager
  - accessing, 2-1
  - copying a template, 2-8
  - creating a Data Definition, 2-2
  - defining the template, 2-5
  - editing templates, 2-10
  - previewing a template, 2-10
  - template mapping, 2-11
  - updating a template, 2-8
  - viewing a template, 2-8
- Template Manager
  - features, 2-1
- template mapping, 2-5, 2-7
- template properties
  - setting, 2-12
- templates
  - copying, 2-8
  - editing in the Template Manager, 2-10
  - updating, 2-10
- temporary directory
  - setting, 4-3
- translatable templates, 2-12
- translations
  - enabling, 2-17
  - incomplete, 2-16
  - progress indicator, 2-16
  - status indicator, 2-16

## **U**

---

- Update Configuration button
  - data definition, 2-5
  - templates, 2-12
- updating a data definition, 2-4
- updating a template, 2-8
- uploading objects, C-1

## **V**

---

- viewing a data definition, 2-4
- viewing a template, 2-8

## **W**

---

- WebDAV delivery, 7-15

## **X**

---

- XDOloader, C-1
- XLIFF file, 2-13
- XML Report Publisher program, 3-1

