# Oracle® XML Publisher

User’s Guide

Release 11*i*

**Part No. B13817-02**

January 2005

ORACLE®

Oracle XML Publisher User’s Guide, Release 11*i*

Part No. B13817-02

Primary Author: Leslie Studdard

Contributor: Nancy Chung, Tim Dexter, Edward Jiang, Incheol Kang, Kei Saito

# Contents

# Send Us Your Comments

**Oracle XML Publisher User's Guide, Release 11*i***

**Part No. B13817-02**

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

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

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

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

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

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

# Preface

## Intended Audience

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

This manual is intended to instruct users on how to use Oracle XML Publisher to create customized reports out of the Oracle E-Business Suite.

This guide assumes you have a working knowledge of the following:

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

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

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

See Related Documents on page  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

## Related Documents

### Online Documentation

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

- **PDF Documentation**- See the Online Documentation CD for current PDF documentation for your product with each release. This Documentation CD is also available on Oracle*MetaLink* and is updated frequently.
- **Online Help** - You can refer to Oracle Applications Help for current HTML online help for your product. Oracle provides patchable online help, which you can apply to your system for updated implementation and end user documentation. No system downtime is required to apply online help.
- **Release Content Document** - See the Release Content Document for descriptions of new features available by release. The Release Content Document is available on Oracle*MetaLink*.
- **About document** - Refer to the About document for information about your release, including feature updates, installation information, and new documentation or documentation patches that you can download. The About document is available on Oracle*MetaLink*.

### Related Guides

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

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

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

### Documents Related to this Product

**Oracle Applications User's Guide**

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

**Oracle Applications System Administrator's Guide**

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

**"About" Document**

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

## Do Not Use Database Tools to Modify Oracle Applications Data

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

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

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

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

# 1
# XML Publisher Introduction

This chapter covers the following topics:

- Introduction
- Process Overview
- Structure of this Manual

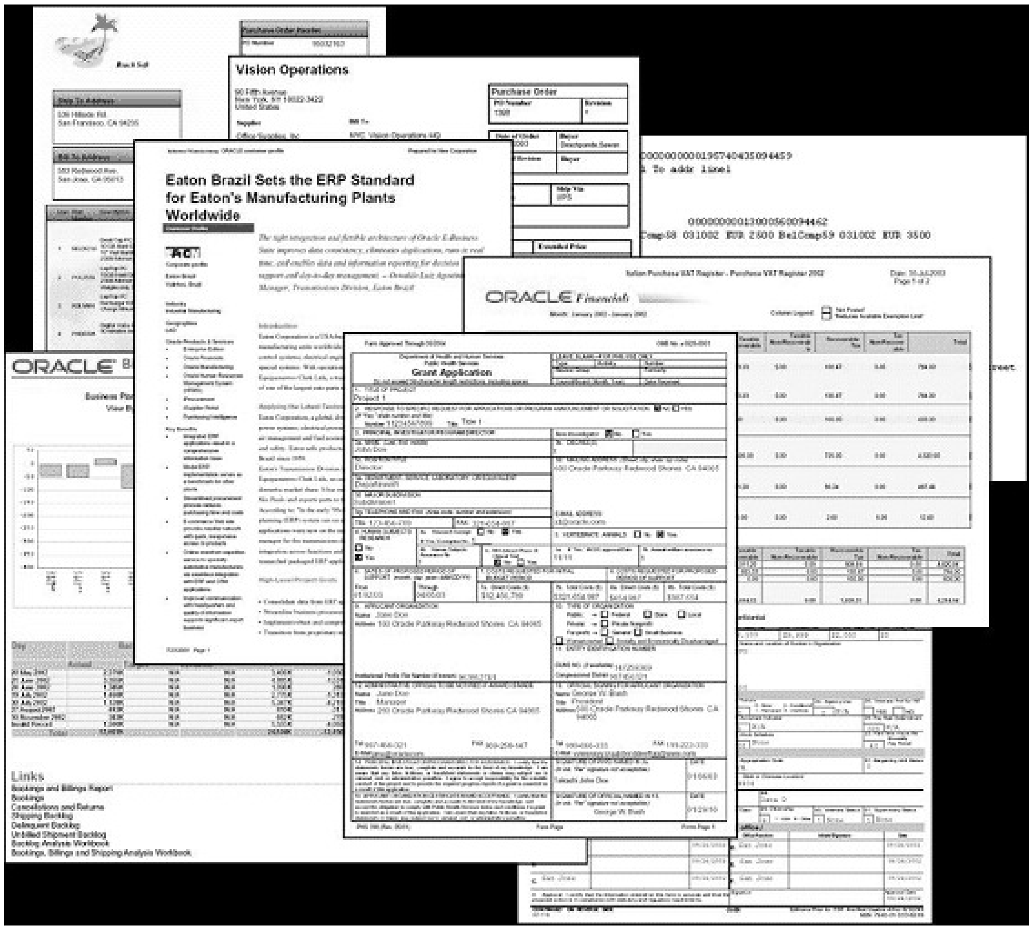
## Introduction

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

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

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

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


Vision Operations
Purchase Order
Eaton Brazil Sets the ERP Standard for Eaton's Manufacturing Plants Worldwide
ORACLE
Grant Application
Links

## User Interfaces

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

> **Note:** The Oracle Application Object Library (fnd) patch 3435480 fully integrates XML Publisher with standard request submission both in Oracle Forms and HTML-based applications. You are no longer required to run the XML Publisher Concurrent Request.

**Template Manager**

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

**XML Report Publisher Concurrent Request**

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

## Process Overview

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

### Design Time

1. Register the E-Business Suite report as a Data Definition in the Template Manager.

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

2. Design your template.

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

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

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

3. Create a Template in the Template Manager for your template design file.

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

### Runtime

1. Set the concurrent program to generate XML.

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

   > **Note:** The Oracle Application Object Library (fnd) patch 3435480 fully integrates XML Publisher with the concurrent manager's standard request submission both in Oracle Forms and HTML-based applications. Simply run the request and select your template from the Submit Request user interface and XML Publisher merges the template and data all in a single step. You are no longer required to run the XML Publisher Concurrent Request.

3. Run the XML Publisher Concurrent Request.

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

## Structure of this Manual

This manual contains the following information to enable you to get started and fully implement the capabilities of XML Publisher.

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 for a PDF

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

Using the Template Manager - describes how to register your Oracle report as a data definition and upload your templates to the Template Manager.

Generating Your Customized Output - describes how to submit your report request to generate output in your customized template.

XML Publisher Extended Functions - describes advanced SQL and XSL functions that XML Publisher has extended for use in templates.

Calling XML Publisher APIs - intended for developers, this section describes how to leverage XML Publisher's processing engines via APIs.

Delivery Manager - intended for developers, this section describes how to use XML Publisher's Delivery Manager APIs to deliver your documents via multiple channels, and how to create a custom channel.

XML Publisher Configuration File - this chapter describes the properties that you can set in the XML Publisher Configuration file.

# 2

# Creating an RTF Template

This chapter covers the following topics:

## Introduction

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

XML Publisher's RTF Template Parser converts documents saved as the RTF file type to XSL-FO. You can therefore create report designs using your standard word processing application's design features and XML Publisher will recognize and maintain the design.

During design time, you add data fields and other markup to your template using XML Publisher's simplified tags for XSL expressions. These tags associate the XML report data to your report layout. If you are familiar with XSL and prefer not to use the simplified tags, XML Publisher also supports the use of pure XSL elements in the template.

In addition to your word processing application's formatting features, XML Publisher supports other advanced design features such as conditional formatting, dynamic data columns, and dynamic table of contents.

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

## Supported Modes

XML Publisher supports two methods for creating RTF templates:

- Basic RTF Method

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

- Form Field Method

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

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

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

## Prerequisites

Before you design your template, you must:

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

## Overview of Creating an RTF Template

Creating an RTF template file consists of two basic steps:

1. Design your template layout.

   Use the formatting features of your word processing application and save the file as RTF.

2. Mark up your template layout.

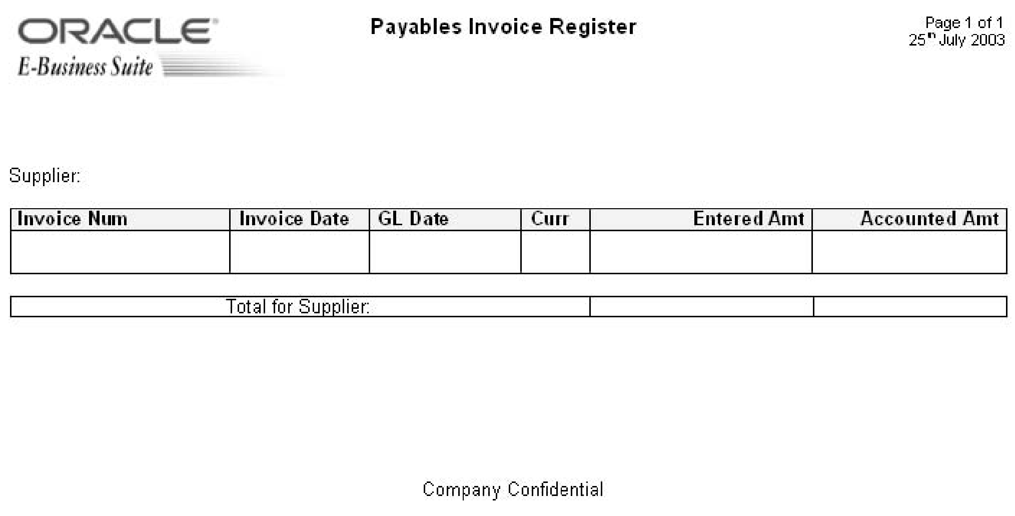
   Insert the XML Publisher simplified tags.

When you design your template layout, you must understand how to associate the XML input file to the layout. The following example presents a sample template layout with its input XML file to illustrate how to make the proper associations to add the markup tags to the template.

### Associating the XML Data to the Template Layout

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

***Sample Template Layout***



Payables Invoice Register

Page 1 of 1
25th July 2003

Supplier:

| Invoice Num | Invoice Date | GL Date | Curr | Entered Amt | Accounted Amt |
|---|---|---|---|---|---|
| | | | | | |

| | | |
|---|---|---|
| Total for Supplier: | | |

Company Confidential

Note the following:

- The data fields that are defined on the template

  For example: Supplier, Invoice Number, and Invoice Date

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

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

## XML Input File

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

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

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

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

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

## Identifying Placeholders and Groups

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

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

> **Note:** XML Publisher supports regrouping of data if your report requires grouping that does not follow the hierarchy of your incoming XML data. For information on using this feature, see Regrouping the XML Data, page 2-32.

### Placeholders

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

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

### Identifying the Groups of Repeating Elements

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

- Fields that repeat for each supplier
- Fields that repeat for each invoice

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

**Suppliers**

- Supplier Name
- Invoices
  - Invoice Num
  - Invoice Date
  - GL Date
  - Currency
  - Entered Amount
  - Accounted Amount
- Total Entered Amount
- Total Accounted Amount

Compare this structure to the hierarchy of the XML input file. The fields that belong to the Suppliers group shown above are children of the element G_VENDOR_NAME. The fields that belong to the Invoices group are children of the element G_INVOICE_NUM.

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

## Designing the Template Layout

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

For example:

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

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

For a detailed list of supported formatting features, see Supported Native Formatting Features, page 2-15.

## Adding Markup to the Template Layout

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

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

XML Publisher provides tags to add markup to your template.

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

### Creating Placeholders

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

Enter placeholders in your document using the following syntax:

**<?***XML element tag name***?>**

> **Note:** The placeholder must match the XML element tag name exactly. It is case sensitive.

There are two ways to insert placeholders in your document:

1. Basic RTF Method: Insert the placeholder syntax directly into your template document.
2. Form Field Method: (Requires Microsoft Word) Insert the placeholder syntax in Microsoft Word's Text Form Field Options window. This method allows you to maintain the appearance of your template.

#### Basic RTF Method

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

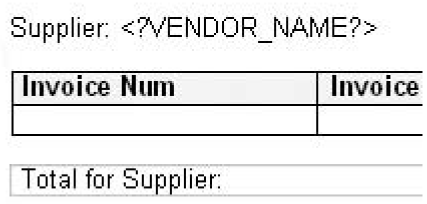
Enter the element's XML tag name using the syntax:

**<?***XML element tag name***?>**

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

**<?VENDOR_NAME?>**

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



Supplier: <?VENDOR_NAME?>

|  |  |
|---|---|
| Invoice Num | Invoice |
|  |  |

Total for Supplier:

## Form Field Method

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

1. Enable the **Forms** toolbar in your Microsoft Word application.
2. Position your cursor in the place you want to create a placeholder.
3. Select the **Text Form Field** toolbar icon. This action inserts a form field area in your document.
4. Double-click the form field area to invoke the **Text Form Field Options** dialog box.
5. (Optional) Enter a description of the field in the **Default text** field. The entry in this field will populate the placeholder's position on the template.

   For the example, enter "Supplier 1".
6. Select the **Add Help Text** button.
7. In the help text entry field, enter the XML element's tag name using the syntax:

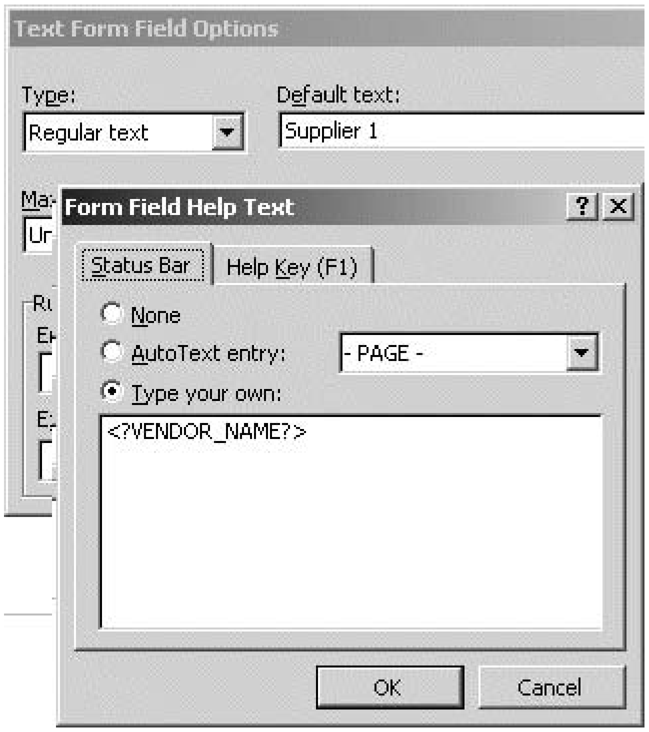
   **<?***XML element tag name***?>**

   You can enter multiple element tag names in the text entry field.

   In the example, the report field "Supplier" maps to the XML element VENDOR_NAME. In the **Form Field Help Text** field enter:

   **<?VENDOR_NAME?>**

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


Text Form Field Options

Type: Regular text

Default text: Supplier 1

Form Field Help Text

Status Bar | Help Key (F1)

None

AutoText entry: - PAGE -

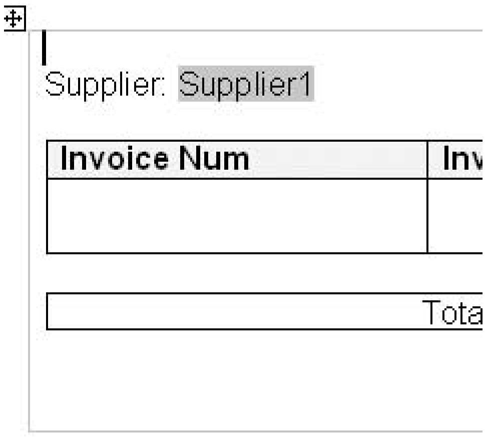
[✓ Type your own:]

<?VENDOR_NAME?>

8. Select **OK** to apply.

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

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


Supplier: Supplier1

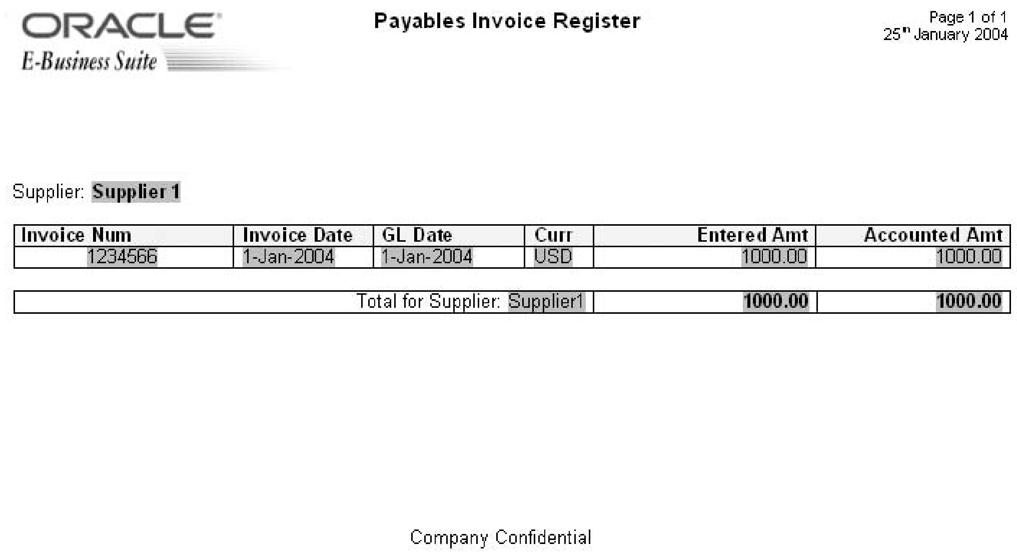
## Complete the Example

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

| Template Field Name | Default Text Entry (Form Field Method) | Placeholder Entry (XML Tag Name) |
|---|---|---|
| Invoice Num | 1234566 | <?INVOICE_NUM?> |
| Invoice Date | 1-Jan-2004 | <?INVOICE_DATE?> |
| GL Date | 1-Jan-2004 | <?GL_DATE?> |
| Curr | USD | <?INVOICE_CURRENCY_ CODE?> |
| Entered Amt | 1000.00 | <?ENT_AMT?> |
| Accounted Amt | 1000.00 | <?ACCTD_AMT?> |
| (Total of Entered Amt column) | 1000.00 | <?ENT_SUM_VENDOR?> |
| (Total of Accounted Amt column) | 1000.00 | <?ACCTD_SUM_VENDOR?> |

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

See the Payables Invoice Register with Completed Basic rtf Markup, page 2-10 for the completed basic rtf markup.



ORACLE E-Business Suite

Payables Invoice Register

Page 1 of 1
25th January 2004

Supplier: Supplier 1

| Invoice Num | Invoice Date | GL Date | Curr | Entered Amt | Accounted Amt |
|---|---|---|---|---|---|
| 1234566 | 1-Jan-2004 | 1-Jan-2004 | USD | 1000.00 | 1000.00 |

| | | |
|---|---|---|
| Total for Supplier: Supplier1 | 1000.00 | 1000.00 |

Company Confidential

## Defining Groups

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

In the example, for each occurrence of G_VENDOR_NAME in the XML file, we want the template to display its child elements VENDOR_NAME (Supplier

Name), G_INVOICE_NUM (the Invoices group), Total Entered Amount, and Total Accounted Amount. And, for each occurrence of G_INVOICE_NUM (Invoices group), we want the template to display Invoice Number, Invoice Date, GL Date, Currency, Entered Amount, and Accounted Amount.

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

Insert the following tag before the first element:

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

Insert the following tag after the final element:

**`<?end for-each?>`**

### Grouping scenarios

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

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

### Basic RTF Method

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

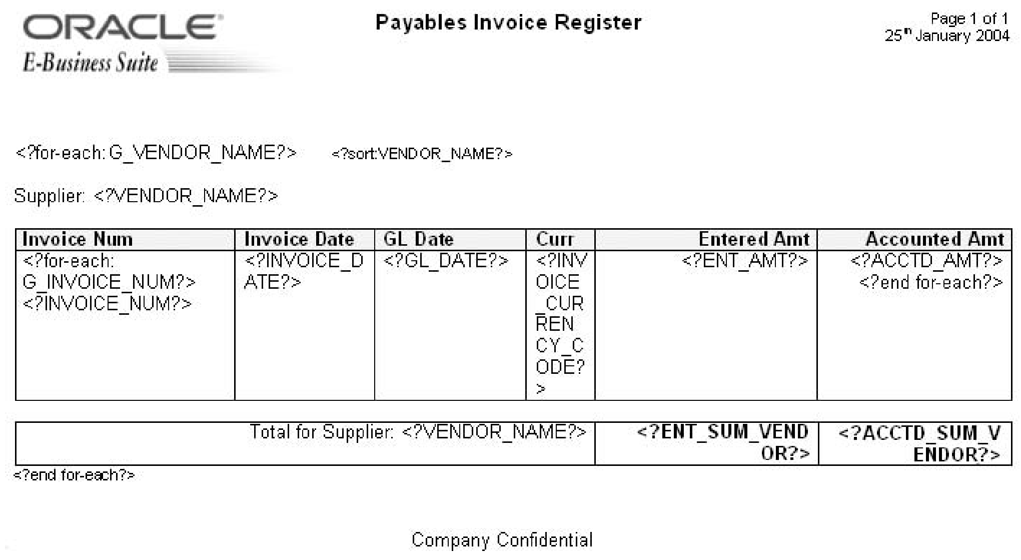
To create the Suppliers group in the example, insert the tag

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

before the Supplier field that you previously created.

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

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


Payables Invoice Register

Page 1 of 1
25th January 2004

<?for-each: G_VENDOR_NAME?> <?sort:VENDOR_NAME?>

Supplier: <?VENDOR_NAME?>

| Invoice Num | Invoice Date | GL Date | Curr | Entered Amt | Accounted Amt |
|---|---|---|---|---|---|
| <?for-each: G_INVOICE_NUM?> <?INVOICE_NUM?> | <?INVOICE_DATE?> | <?GL_DATE?> | <?INVOICE_CURRENCY_CODE?> | <?ENT_AMT?> | <?ACCTD_AMT?> <?end for-each?> |

| | | |
|---|---|---|
| Total for Supplier: <?VENDOR_NAME?> | <?ENT_SUM_VENDOR?> | <?ACCTD_SUM_VENDOR?> |

<?end for-each?>

Company Confidential

## Form Field Method

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

   In the help text field enter:

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

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

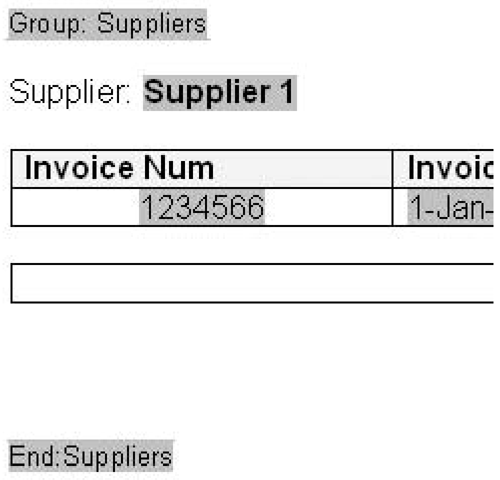
   **<?for-each:G_VENDOR_NAME?>**

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

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

   For the example, enter the Default text "End: Suppliers" after the summary row to designate the end of the group on the template.

   The following figure shows the template after the markup to designate the Suppliers group was added.


Group: Suppliers

Supplier: Supplier 1

| Invoice Num | Invoic |
|---|---|
| 1234566 | 1-Jan- |

End:Suppliers

## Complete the Example

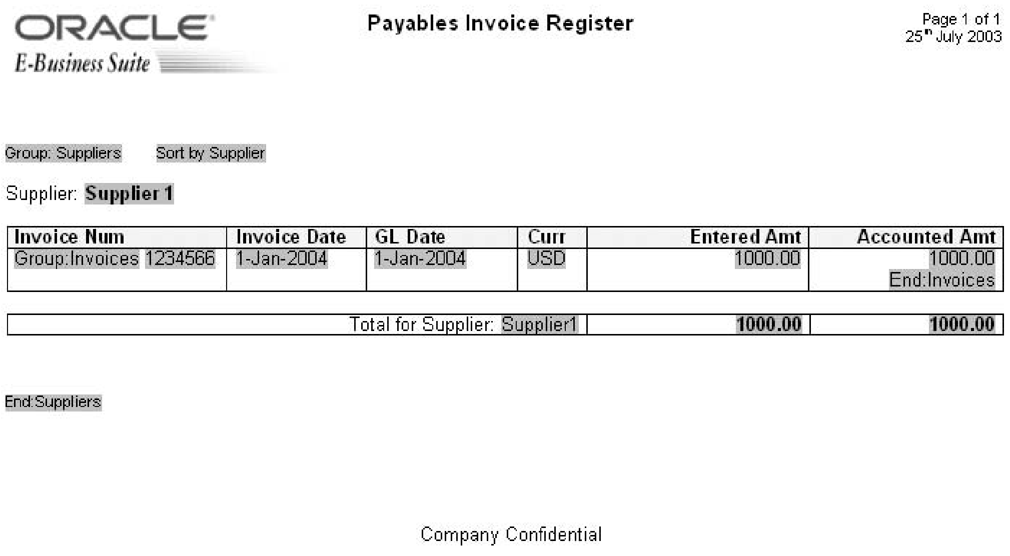
The second group in the example is the invoices group. The repeating elements in this group are displayed in the table. For each invoice, the table row should repeat. Create a group within the table to contain these elements.

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

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

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

The following figure shows the completed example using the form field method:



ORACLE E-Business Suite | Payables Invoice Register | Page 1 of 1 25th July 2003

Group: Suppliers Sort by Supplier

Supplier: Supplier 1

| Invoice Num | Invoice Date | GL Date | Curr | Entered Amt | Accounted Amt |
|---|---|---|---|---|---|
| Group:Invoices 1234566 | 1-Jan-2004 | 1-Jan-2004 | USD | 1000.00 | 1000.00 End:Invoices |

| Total for Supplier: Supplier1 | 1000.00 | 1000.00 |
|---|---|---|

End:Suppliers

Company Confidential

## Defining Headers and Footers

XML Publisher supports the use of the native RTF header and footer feature. To create a header or footer, use the your word processing application's header and footer insertion tools.

### Multiple Headers and Footers

If your template requires multiple headers and footers, create them by using XML Publisher tags to define the body area of your report. When you define the body area, the elements occurring before the beginning of the body area will compose the header. The elements occurring after the body area will compose the footer.

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

**`<?start:body?>`**

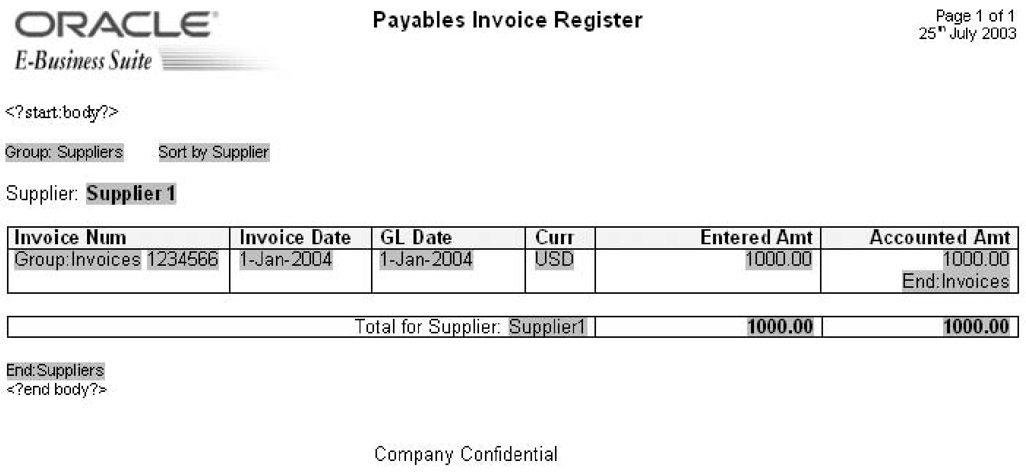
**`<?end body?>`**

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

The Payables Invoice Register contains a simple header and footer and therefore does not require the start body/end body tags. However, if you wanted to add another header to the template, define the body area as follows:

1. Insert **`<?start:body?>`** before the Suppliers group tag: **`<?for-each:G_VENDOR_NAME?>`**
2. Insert **`<?end body?>`** after the Suppliers group closing tag: **`<?end for-each?>`**

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


ORACLE E-Business Suite

Payables Invoice Register

Page 1 of 1
25th July 2003

<?start:body?>

Group: Suppliers    Sort by Supplier

Supplier: **Supplier 1**

| Invoice Num | Invoice Date | GL Date | Curr | Entered Amt | Accounted Amt |
|---|---|---|---|---|---|
| Group:Invoices 1234566 | 1-Jan-2004 | 1-Jan-2004 | USD | 1000.00 | 1000.00 End:Invoices |

| Total for Supplier: Supplier1 | **1000.00** | **1000.00** |
|---|---|---|

End:Suppliers
<?end body?>

Company Confidential

### Inserting Placeholders in the Header and Footer

At the time of this writing, Microsoft Word does not support form fields in the header and footer. You must therefore insert the placeholder syntax directly into the template (basic RTF method).

## Including Images

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

- Direct Insertion

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

- URL Reference

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

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

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

- OA Media Directory Reference

  1. Insert a dummy image in your template.
  2. In the **Format Picture** dialog box select the **Web** tab. Enter the following syntax in the **Alternative text** region to reference the OA_MEDIA directory:

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

     For example, enter:

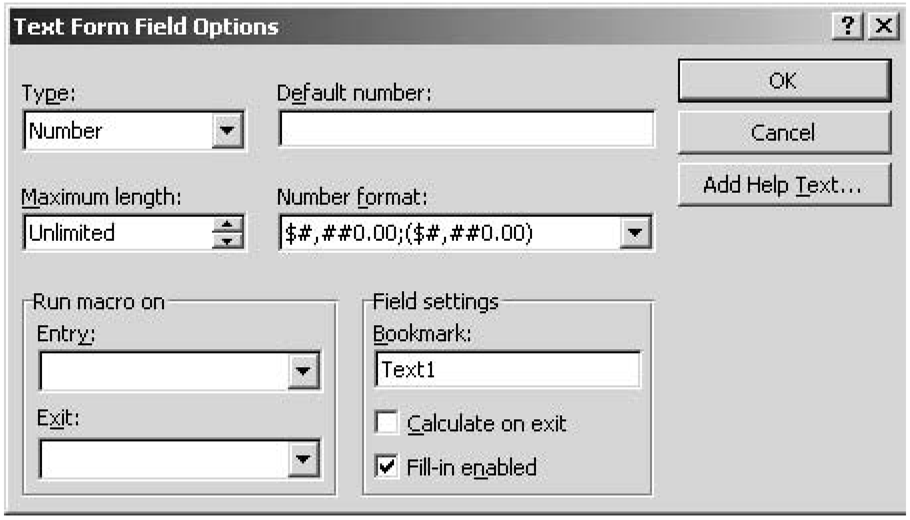
     **`url:{'${OA_MEDIA}/ORACLE_LOGO.gif'}`**

## Supported Native Formatting Features

In addition to the features already listed, XML Publisher supports the following features of your word processing application.

### Number Formatting

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



Type: Number

Default number:

Maximum length: Unlimited

Number format: $#,##0.00;($#,##0.00)

Bookmark: Text1

[✓ Fill-in enabled]

To apply a number format to a form field:

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

At runtime the numeric values from your XML data file will be formatted according to the field properties.

> **Note:** This function will only work if your data contains raw numbers, such as 1000.00. If the number has been formatted for European countries (for example: 1.000,00) the format will not work.

You can also use the native XSL format-number function to format numbers. See: Native XSL Number Formatting, page 2-50.

### Date Formatting

Microsoft Word's native date formatting feature is supported as well for specific XML schema date formats (see the Note below).

To apply a date format to a form field:

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

**Important:** This function will only work on the XML schema date format: YYY-MM-DDThh:mm:ss+HH:MM, where:

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

An example of this construction is: 2005-01-01T09:30:10-07:00. The data after the "T" is optional, therefore the following date: 2005-01-01 can be formatted using Microsoft Word's native date formatting.

To show a timestamp component in your date field you must provide the complete XML schema date format including the time zone offset. If you do not include the time zone offset, the time will be formatted to the UTC time.

## General Features

- Large blocks of text
- Page breaks

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

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

  At runtime each Supplier will start on a new page.
- Page numbering

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

  1. From the **Insert** menu, select **Page Numbers...**
  2. Select the **Position**, **Alignment**, and **Format** as desired.

  At runtime the page numbers will be displayed as selected.

## Alignment

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

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

## Tables

Supported table features include:

- Nested Tables
- Cell Alignment

  You can align any object in your template using your word processing application's alignment tools. This alignment will be reflected in the final report output.
- Row spanning and column spanning

  You can span both columns and rows in your template as follows:

  1. Select the cells you wish to merge.
  2. From the **Table** menu, select **Merge Cells**.
  3. Align the data within the merged cell as you would normally.

  At runtime the cells will appear merged.
- Table Autoformatting

  XML Publisher recognizes the table autoformats available in Microsoft Word.

  1. Select the table you wish to format.
  2. From the **Table** menu, select **Autoformat**.
  3. Select the desired table format.

  At runtime, the table will be formatted using your selection.
- Cell patterns and colors

  You can highlight cells or rows of a table with a pattern or color.

  1. Select the cell(s) or table.
  2. From the **Table** menu, select **Table Properties**.
  3. From the **Table** tab, select the **Borders and Shading...** button.
  4. Add borders and shading as desired.
- Repeating table headers

  If your data is displayed in a table, and you expect the table to extend across multiple pages, you can define the header rows that you want to repeat at the start of each page.

  1. Select the row(s) you wish to repeat on each page.
  2. From the **Table** menu, select **Heading Rows Repeat**.

## Date Fields

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

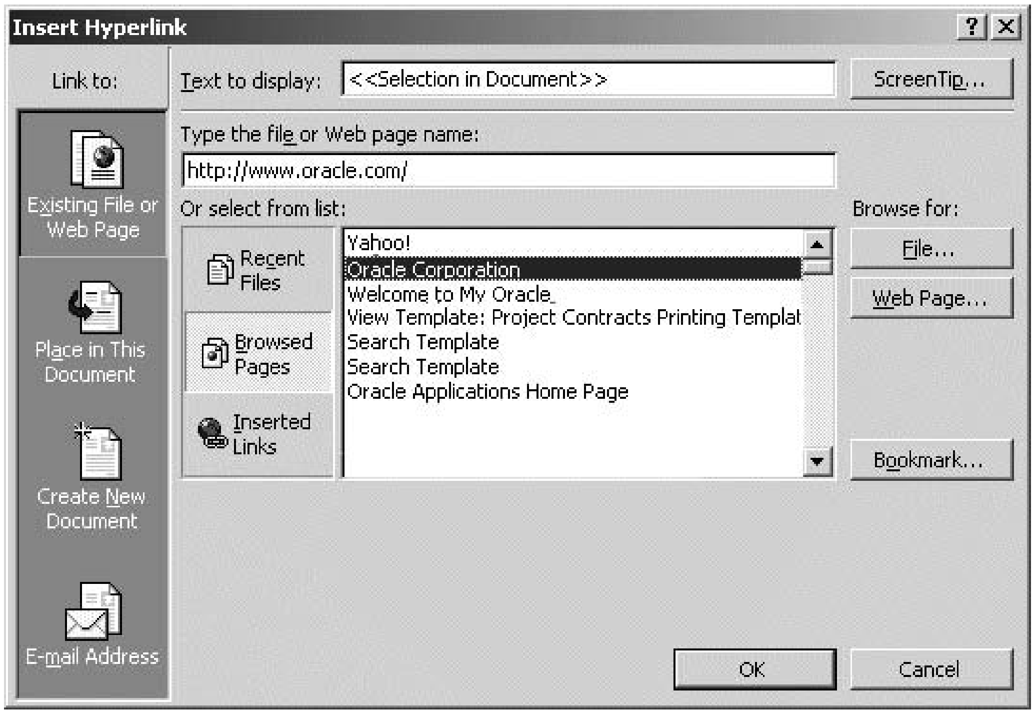
## Special Features

### Embedded Hyperlinks

You can add fixed or dynamic hyperlinks to your template.

- To insert Static Hyperlinks, use your word processing application's insert hyperlink feature.

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



Text to display: <<Selection in Document>>

Type the file or Web page name: http://www.oracle.com/

Or select from list:
Yahoo!
Oracle Corporation
Welcome to My Oracle
View Template: Project Contracts Printing Templat
Search Template
Search Template
Oracle Applications Home Page

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

  ```
  {URL_LINK}
  ```

  where URL_LINK is the incoming data element name.

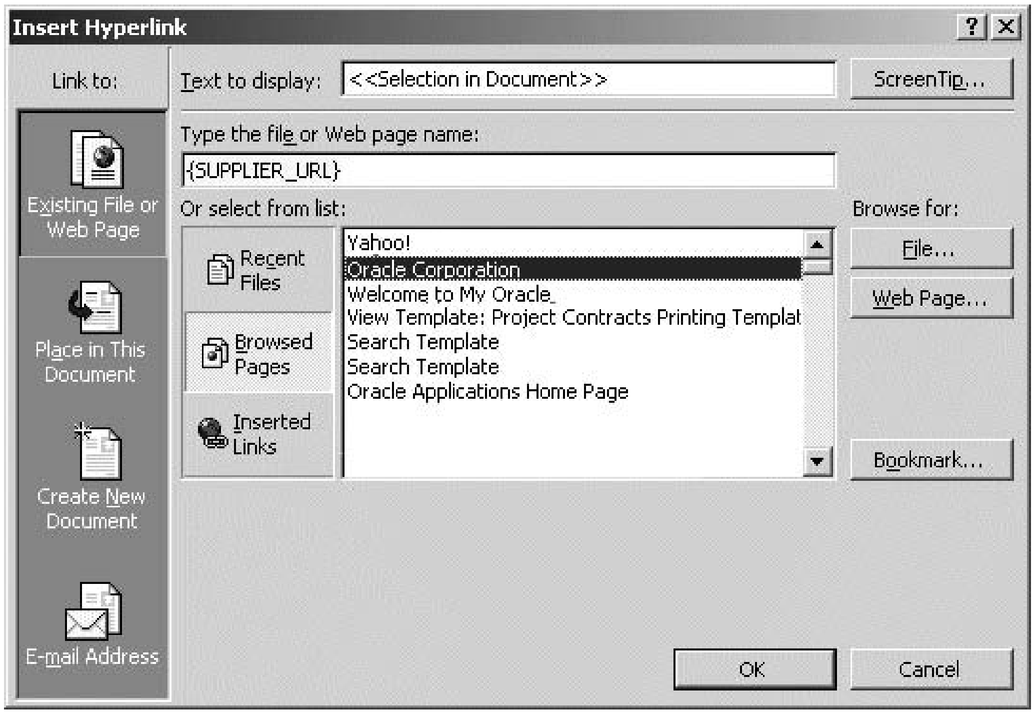
  If you have a fixed URL that you want to pass parameters to, enter the following syntax:

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

  where PRODUCT_NAME is the incoming data element name.

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

  The following figure shows the insertion of a dynamic hyperlink using Microsoft Word's **Insert Hyperlink** dialog box. The data element SUPPLIER_URL from the incoming XML file will contain the hyperlink that will be inserted into the report at runtime.



Text to display: <<Selection in Document>>

Type the file or Web page name: {SUPPLIER_URL}

Or select from list:

Yahoo!
Oracle Corporation
Welcome to My Oracle
View Template: Project Contracts Printing Templat
Search Template
Search Template
Oracle Applications Home Page

## Table of Contents/Dynamic TOC

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

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

To create dynamic headings:

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

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

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

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

## Namespace Support

If your XML data contains namespaces, you must declare them in the template prior to referencing the namespace in a placeholder. Declare the namespace in the template using either the basic RTF method or in a form field. Enter the following syntax:

**`<?namespace:`***`namespace name`***`=`** *`namespace url`***`?>`**

For example:

**<?namespace:fsg=http://www.oracle.com/fsg/2002-30-20/?>**

Once declared, you can use the namespace in the placeholder markup, for example: **<?fsg:ReportName?>**

## Dynamic Data Columns

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

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

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

- Dynamic Column Header

  **<?split-column-header:***group element name***?>**

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

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

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

- Dynamic Column Width

  **<?split-column-width:***name***?>** or

  **<?split-column-width:@width?>**

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

  - An XML element stores the value of the width. In this case, use the syntax **<?split-column-width:***name***?>**, where *name* is the XML element tag name that contains the value for the width.
  - If the element defined in the split-column-header tag, contains a width attribute, use the syntax **<?split-column-width:@width?>** to use the value of that attribute.

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

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

| **Width Definition** | **Column 1 (Width = 10)** | **Column 2 (Width = 12)** | **Column 3 (Width = 14)** |
|---|---|---|---|
| Multiplier not present -% width | 10/10+12+14*100 28% | %Width = 33% | %Width =39% |
| Multiplier = 6 - width | 60 pts | 72 pts | 84 pts |

### Horizontal table break with row header number

**`<?horizontal-break-table:`*`number`*`?>`**

If columns exceed one page, this tag allows you to specify how many row heading columns will repeat on subsequent pages with the continuing columns.

### Example of Dynamic Data Columns

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

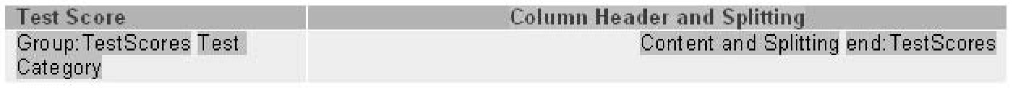
| **Test Score** | **Test Score Range 1** | **Test Score Range 2** | **Test Score Range 3** | ***...Test Score Range n*** |
|---|---|---|---|---|
| Test Category | # students in Range 1 | # students in Range 2 | # students in Range 3 | # of students in Range *n* |

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

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

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

Using the dynamic column tags in form fields, set up the table in two columns as shown in the following figure. The first column, "Test Score" is static. The second column, "Column Header and Splitting" is the dynamic column. At runtime this column will split according to the data, and the header for each column will be appropriately populated. The Default Text entry and Form Field Help entry for each field are listed in the table following the figure. (See Form Field Method, page 2- 7 for more information on using form fields).



| Test Score | Column Header and Splitting |
|---|---|
| Group:TestScores Test Category | Content and Splitting end:TestScores |

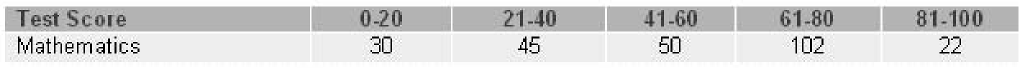
| Default Text Entry | Form Field Help Text Entry |
|---|---|
| Group:TestScores | <?for-each:TestScores?> |
| Test Category | <?TestCategory?> |
| Column Header and Splitting | <?split-column-header:TestScore?> <?split-column-width:@width?> <?TestScoreRange?>% |
| Content and Splitting | <?split-column-data:TestScore?>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

- **Test Score** is the boilerplate column heading.

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

  > **Note:** If the tag (**`<?split-column-width-unit:`*`value`*`?>`**) were present, then the columns would have a specific width in points. If the total column widths were wider than the allotted space on the page, then the table would break onto another page.
  >
  > The "horizontal-break-table" tag could then be used to specify how many columns to repeat on the subsequent page. For example, a value of "1" would repeat the column "Test Score" on the subsequent page, with the continuation of the columns that did not fit on the first page.

The template will render the following result in the output:



| Test Score | 0-20 | 21-40 | 41-60 | 61-80 | 81-100 |
|---|---|---|---|---|---|
| Mathematics | 30 | 45 | 50 | 102 | 22 |

## Data Reporting Features

XML Publisher provides commands that allow you to control what data is reported at publishing time. Not only can you customize the report by selecting the fields to display, but you can also set up parameters within the template to define what is reported (from the data source) and how it is sorted.

> **Note:** For the XSL equivalents of the following, see XSL Equivalent Syntax, page 7- 4 .

### Sorting Fields

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

**`<?sort:`*`element name`*`?>`**

To sort the example by Supplier (VENDOR_NAME), enter the following after the **`<?for-each:G_VENDOR_NAME?>`** tag:

**`<?sort:VENDOR_NAME?>`**

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

**`<?sort:VENDOR_NAME?> <?sort:INVOICE_NUM?>`**

## Applying Conditional Formatting

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

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

### If Statements

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

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

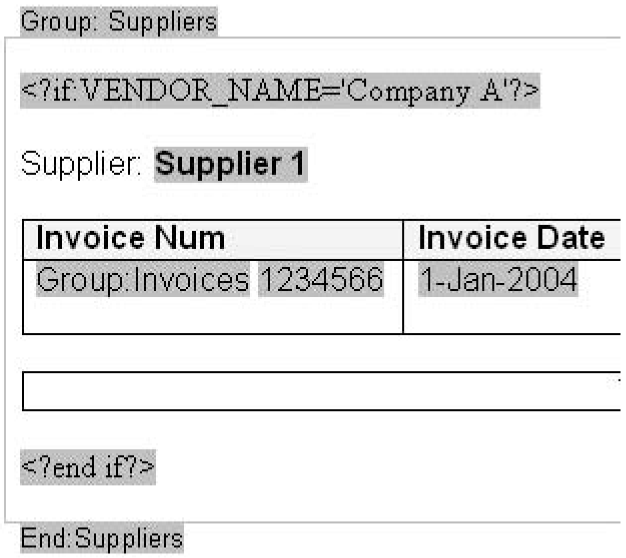
   **`<?if:`*`condition`*`?>`**

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

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

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

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



Group: Suppliers

<?if:VENDOR_NAME='Company A'?>

Supplier: **Supplier 1**

| Invoice Num | Invoice Date |
|---|---|
| Group:Invoices 1234566 | 1-Jan-2004 |

<?end if?>

End:Suppliers

### Choose Statements

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

Use the following syntax for these elements:

**`<?choose?>`**

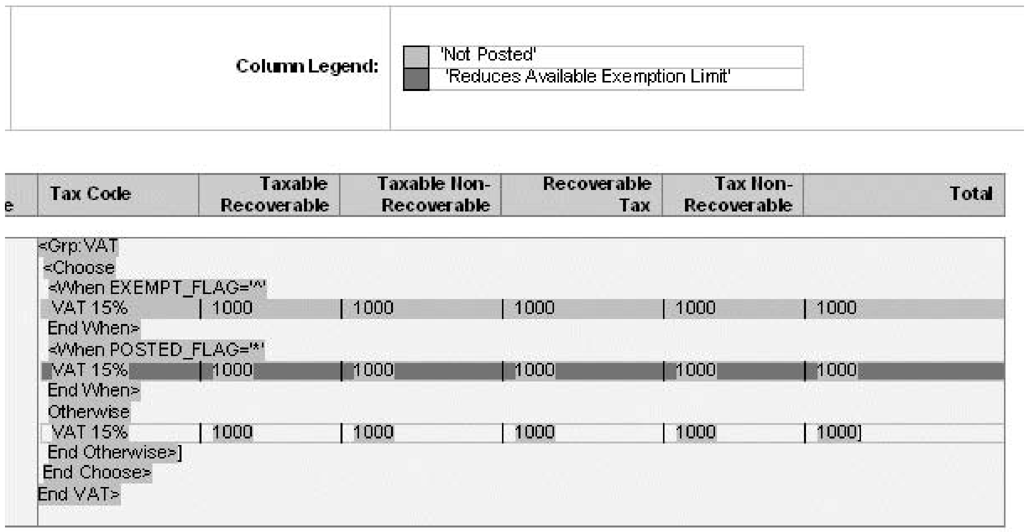
**`<?when:`*`expression`*`?>`**

**`<?otherwise?>`**

## "Choose" Conditional Formatting Example

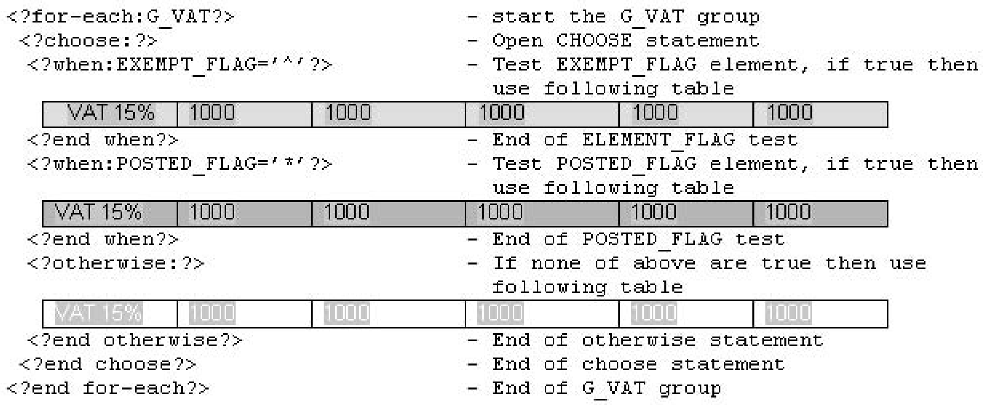
This example shows a **`choose`** expression in which the display of a row of data depends on the value of the fields EXEMPT_FLAG and POSTED_FLAG. When the EXEMPT_FLAG equals "^", the row of data will render light gray. When POSTED_FLAG equals "*" the row of data will render shaded dark gray. Otherwise, the row of data will render with no shading.

In the following figure, the form field default text is displayed. The form field help text entries are shown in the table following the example.



| Column Legend: | |
|---|---|
| ▨ | 'Not Posted' |
| ■ | 'Reduces Available Exemption Limit' |

| e | Tax Code | Taxable Recoverable | Taxable Non-Recoverable | Recoverable Tax | Tax Non-Recoverable | Total |
|---|---|---|---|---|---|---|
| | <Grp:VAT | | | | | |
| | <Choose | | | | | |
| | <When EXEMPT_FLAG='^' | | | | | |
| | VAT 15% | 1000 | 1000 | 1000 | 1000 | 1000 |
| | End When> | | | | | |
| | <When POSTED_FLAG='*' | | | | | |
| | VAT 15% | 1000 | 1000 | 1000 | 1000 | 1000 |
| | End When> | | | | | |
| | Otherwise | | | | | |
| | VAT 15% | 1000 | 1000 | 1000 | 1000 | 1000] |
| | End Otherwise>] | | | | | |
| | End Choose> | | | | | |
| | End VAT> | | | | | |



```
<?for-each:G_VAT?>                    - start the G_VAT group
 <?choose:?>                          - Open CHOOSE statement
  <?when:EXEMPT_FLAG='^'?>            - Test EXEMPT_FLAG element, if true then
                                        use following table
   | VAT 15% | 1000 | 1000 | 1000 | 1000 | 1000 |
  <?end when?>                        - End of ELEMENT_FLAG test
  <?when:POSTED_FLAG='*'?>            - Test POSTED_FLAG element, if true then
                                        use following table
   | VAT 15% | 1000 | 1000 | 1000 | 1000 | 1000 |
  <?end when?>                        - End of POSTED_FLAG test
  <?otherwise:?>                      - If none of above are true then use
                                        following table
   | VAT 15% | 1000 | 1000 | 1000 | 1000 | 1000 |
  <?end otherwise?>                   - End of otherwise statement
 <?end choose?>                       - End of choose statement
<?end for-each?>                      - End of G_VAT group
```

| Default Text Entry in Example Form Field | Help Text Entry in Form Field |
|---|---|
| <Grp:VAT | <?for-each:VAT?> |
| <Choose | <?choose?> |
| <When EXEMPT_FLAG='^' | <?When EXEMPT_FLAG='^'?> |
| End When> | <?end When?> |
| <When EXEMPT_FLAG='^' | <?When EXEMPT_FLAG='^'?> |
| End When> | <?end When?> |

## Inserting Page Totals

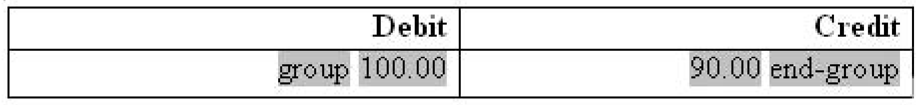
XML Publisher supports page totaling so that you can define fields in your template that at runtime will calculate and display total figures for fields that are displayed on that particular page. Once you define total fields, you can also perform additional functions on the data in those fields.

The following example shows how to set up page total fields in a template to display total credits and debits that have displayed on the page, and then calculate the net of the two fields.

This example uses the following XML:

```
<balance_sheet>
 <transaction>
  <debit>100</debit>
  <credit>90</credit>
 </transaction>
 <transaction>
  <debit>110</debit>
  <credit>80</credit>
 </transaction>
…
<\balance_sheet>
```

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



| Debit | Credit |
|---|---|
| group 100.00 | 90.00 end-group |

You must declare a page total for each element in your report that requires a page total. This takes the form:

**<?add-page-total:***name***;'***element***'?>**
where

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

**'element'** is the XML element field to be totaled

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

| Default Text Entry | Form Field Help Text Entry | Description |
|---|---|---|
| group | **<?for-each:transaction?>** | This field defines the opening "for-each" loop for the **transaction** group. |
| 100.00 | **<?debit?><?add-page-total: dt;'debit'?>** | This field is the placeholder for the **debit** element from the XML file. Because we want to total this field by page, the page total declaration syntax is added. The field defined to hold the total for the **debit** element is **dt**. |
| 90.00 | **<?credit?> <?add-page-total:ct;'credit'?> <add-page-total:net;'debit - credit'?>** | This field is the placeholder for the **credit** element from the XML file. Because we want to total this field by page, the page total declaration syntax is added. The field defined to hold the total for the **credit** element is **ct**. This field also contains a net value declaration. |
| end-group | **<?end for-each?>** | Closes the for-each loop. |

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

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

**<?show-page-total:***name***;'***number-format***'?>**
where

**name** is the name you assigned to give the page total field above and

**number-format** is the format you wish to use to for the display.

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

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

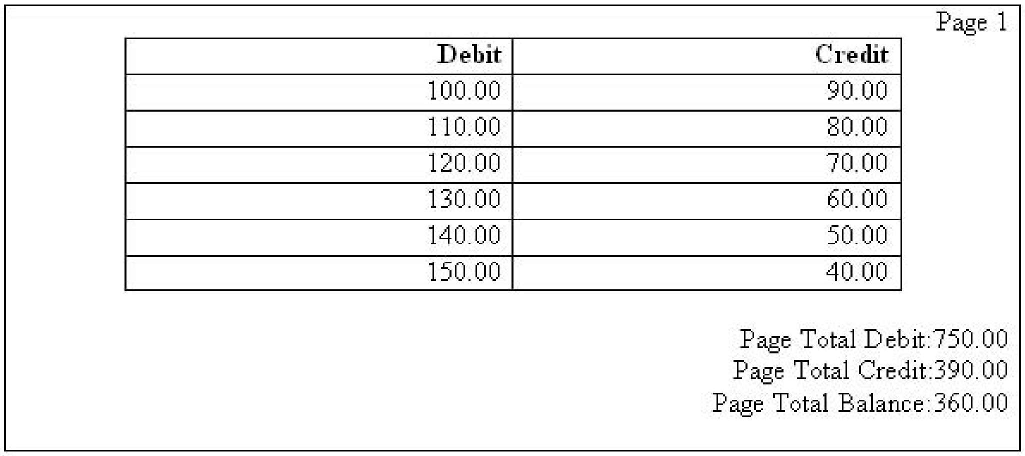
Therefore to complete the example, place the following at the bottom of the template page, or in the footer:

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

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

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

The output for this report is shown in the following graphic:



Page 1

| Debit | Credit |
|---|---|
| 100.00 | 90.00 |
| 110.00 | 80.00 |
| 120.00 | 70.00 |
| 130.00 | 60.00 |
| 140.00 | 50.00 |
| 150.00 | 40.00 |

Page Total Debit:750.00
Page Total Credit:390.00
Page Total Balance:360.00

Note that this page totaling function will only work if your source XML has raw numeric values. The numbers must not be preformatted.

## Conditional Column Formatting

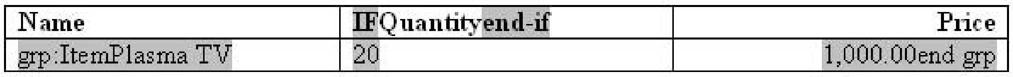
You can conditionally show and hide columns of data in your document output. The following example demonstrates how to setup a table so that a column is only displayed based on the value of an element attribute.

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

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

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

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



| Name | IFQuantityend-if | Price |
|---|---|---|
| grp:ItemPlasma TV | 20 | 1,000.00end grp |

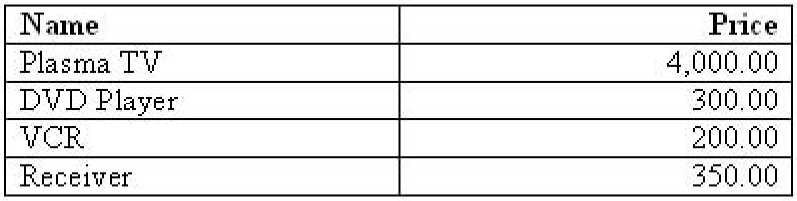
The following table shows the entries made in the template for the example:

| Default Text | Form Field Entry | Description |
|---|---|---|
| grp:Item | **`<?for-each:item?>`** | Holds the opening for-each loop for the **`item`** element. |
| Plasma TV | **`<?name?>`** | The placeholder for the **`name`** element from the XML file. |
| IF Quantity end if | **`<?if@column: /items/ @type="PRIVATE"? >Quantity<?end if?>`** | The "Quantity" column header surrounded by the "if" statement. |
| 20 | **`<?if@column: /items/ @type="PRIVATE"?><? quantity?><?end if?>`** | The placeholder for the quantity element surrounded by the "if" statement. |
| 1,000.00 | **`<?price?>`** | The placeholder for the **`price`** element. |
| end grp | **`<?end for-each?>`** | Closing tag of the for-each loop. |

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

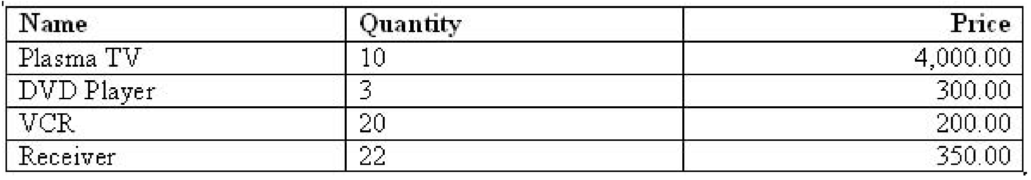
If you did not include the **`@column`** the data would not display in your report as a result of the if statement, but the column still would because you had drawn it in your template.

The example will render the following output:



| Name | Price |
|---|---|
| Plasma TV | 4,000.00 |
| DVD Player | 300.00 |
| VCR | 200.00 |
| Receiver | 350.00 |

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



| Name | Quantity | Price |
|---|---|---|
| Plasma TV | 10 | 4,000.00 |
| DVD Player | 3 | 300.00 |
| VCR | 20 | 200.00 |
| Receiver | 22 | 350.00 |

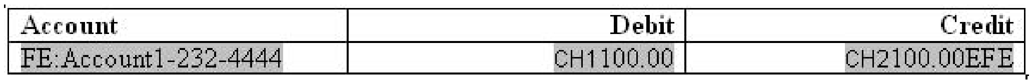
## Conditional Cell Highlighting

You can conditionally highlight individual cells, columns, or rows in your final output. The following example demonstrates how to conditionally highlight a cell based on a value in the XML file.

For this example we will use the following XML:

```
<accounts>
 <account>
  <number>1-100-3333</number>
  <debit>100</debit>
  <credit>300</credit>
 </account>
 <account>
  <number>1-101-3533</number>
  <debit>220</debit>
  <credit>30</credit>
 </account>
 <account>
  <number>1-130-3343</number>
  <debit>240</debit>
  <credit>1100</credit>
 </account>
 <account>
  <number>1-153-3033</number>
  <debit>3000</debit>
  <credit>300</credit>
 </account>
</accounts>
```

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



| Account | Debit | Credit |
|---|---|---|
| FE:Account1-232-4444 | CH1100.00 | CH2100.00EFE |

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

| Default Text Entry | Form Field Entry | Description |
|---|---|---|
| FE:Account | **`<?for-each:account?>`** | Opens the for each-loop for the element **`account`**. |
| 1-232-4444 | **`<?number?>`** | The placeholder for the **`number`** element from the XML file. |
| CH1 | **`<?if:debit>1000?><xsl: attribute xdofo:ctx= "block" name="background-color">red</xsl: attribute><?end if?>`** | This field holds the code to highlight the cell red if the debit amount is greater than 1000. |
| 100.00 | **`<?debit?>`** | The placeholder for the **`debit`** element. |
| CH2 | **`<?if:credit>1000?><xsl: attribute xdofo:ctx= "block" name="background-color">red</xsl: attribute><?end if?>`** | This field holds the code to highlight the cell red if the credit amount is greater than 1000. |
| 100.00 | **`<?credit?>`** | The placeholder for the **`credit`** element. |
| EFE | **`<?end for-each?>`** | Closes the for-each loop. |

The highlighting code for the debit column as shown in the table is:

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

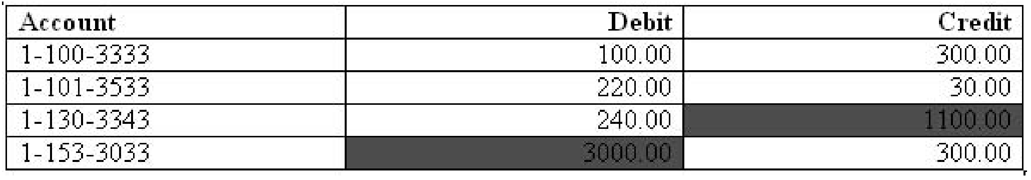
The "if" statement is testing if the debit value is greater than 1000. If it is, then the next lines are invoked. Notice that the example embeds native XSL code inside the "if" statement.

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

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

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

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



| Account | Debit | Credit |
|---|---|---|
| 1-100-3333 | 100.00 | 300.00 |
| 1-101-3533 | 220.00 | 30.00 |
| 1-130-3343 | 240.00 | 1100.00 |
| 1-153-3033 | 3000.00 | 300.00 |

## Regrouping the XML Data

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

### XML Sample

To demonstrate the for-each-group standard, the following XML data sample of a CD catalog listing will be regrouped in a template:

```
<CATALOG>
   <CD>
      <TITLE>Empire Burlesque</TITLE>
      <ARTIST>Bob Dylan</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Columbia</COMPANY>
      <PRICE>10.90</PRICE>
      <YEAR>1985</YEAR>
   </CD>
   <CD>
      <TITLE>Hide Your Heart</TITLE>
      <ARTIST>Bonnie Tylor</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>CBS Records</COMPANY>
      <PRICE>9.90</PRICE>
      <YEAR>1988</YEAR>
   </CD>
   <CD>
      <TITLE>Still got the blues</TITLE>
      <ARTIST>Gary More</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>Virgin Records</COMPANY>
      <PRICE>10.20</PRICE>
      <YEAR>1990</YEAR>
   </CD>
   <CD>
      <TITLE>This is US</TITLE>
      <ARTIST>Gary Lee</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>Virgin Records</COMPANY>
      <PRICE>12.20</PRICE>
      <YEAR>1990</YEAR>
   </CD>
```

Using the regrouping syntax, you can create a report of this data that groups the CDs by country and then by year. You are not limited by the data structure presented.

### Regrouping Syntax

To regroup the data, use the following syntax:

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

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

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

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

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

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

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

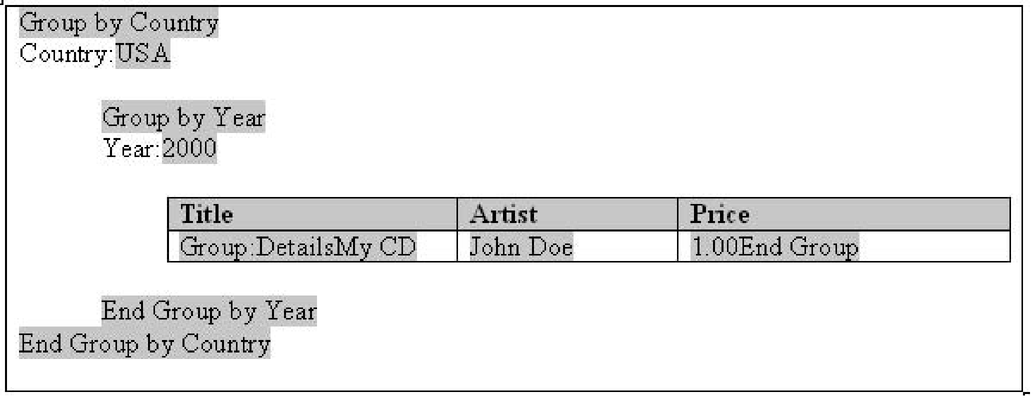
**<?for-each:current-group();YEAR?>**

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

> **Note:** This syntax is a simplification of the XSL for-each-group syntax. If you choose not to use the simplified syntax above, you can use the XSL syntax as shown below. The XSL syntax can only be used within a form field of the template.
>
> ```
> <xsl:for-each-group
>   select=expression
>   group-by="string expression"
>   group-adjacent="string expression"
>   group-starting-with=pattern>
>   <!--Content: (xsl:sort*, content-constructor) -->
> </xsl:for-each-group>
> ```

### Template Example

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



Group by Country
Country:USA

Group by Year
Year:2000

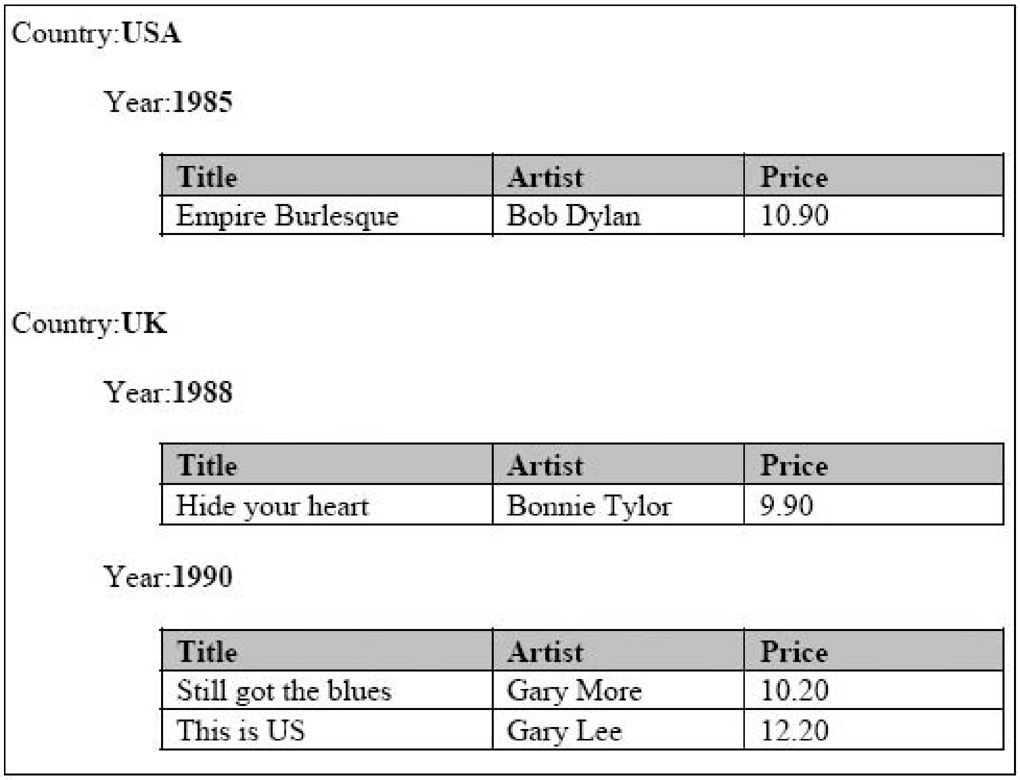
| Title | Artist | Price |
|---|---|---|
| Group:DetailsMy CD | John Doe | 1.00End Group |

End Group by Year
End Group by Country

The following table shows the XML Publisher syntax entries made in the form fields of the preceding template:

| Default Text Entry | Form Field Help Text Entry | Description |
|---|---|---|
| Group by Country | **<?for-each-group: CD;COUNTRY?>** | The **<?for-each-group: CD;COUNTRY?>** tag declares the new group. It regroups the existing CD group by the COUNTRY element. |
| USA | **<?COUNTRY?>** | Placeholder to display the data value of the COUNTRY tag. |
| Group by Year | **<?for-each-group: current-group();YEAR?>** | The **<?for-each-group: current-group();YEAR?>** tag regroups the current group (that is, COUNTRY), by the YEAR element. |
| 2000 | **<?YEAR?>** | Placeholder to display the data value of the YEAR tag. |
| Group: Details | **<?for-each:current-group()?>** | Once the data is grouped by COUNTRY and then by YEAR, the **<?for-each: current-group()?>** command is used to loop through the elements of the current group (that is, YEAR) and render the data values (TITLE, ARTIST, and PRICE) in the table. |
| My CD | **<?TITLE?>** | Placeholder to display the data value of the TITLE tag. |
| John Doe | **<?ARTIST?>** | Placeholder to display the data value of the ARTIST tag. |
| 1.00 | **<?PRICE?>** | Placeholder to display the data value of the PRICE tag. |
| End Group | **<?end for-each?>** | Closes out the **<?for-each: current-group()?>** tag. |
| End Group by Year | **<?end for-each-group?>** | Closes out the **<?for-each-group:current-group();YEAR?>** tag. |
| End Group by Country | **<?end for-each-group?>** | Closes out the **<?for-each-group: CD;COUNTRY?>** tag. |

This template produces the following output when merged with the XML file:



Country:**USA**

Year:**1985**

| **Title** | **Artist** | **Price** |
|---|---|---|
| Empire Burlesque | Bob Dylan | 10.90 |

Country:**UK**

Year:**1988**

| **Title** | **Artist** | **Price** |
|---|---|---|
| Hide your heart | Bonnie Tylor | 9.90 |

Year:**1990**

| **Title** | **Artist** | **Price** |
|---|---|---|
| Still got the blues | Gary More | 10.20 |
| This is US | Gary Lee | 12.20 |

### Regrouping by an Expression

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

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

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

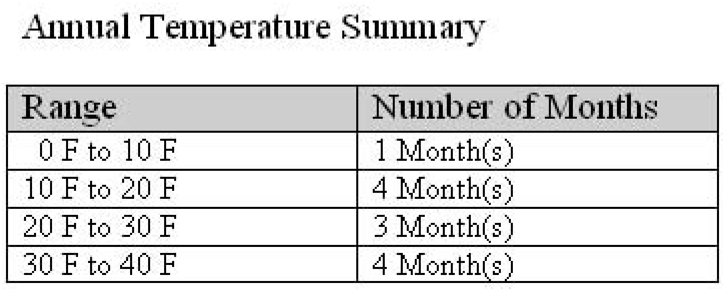
**Example**

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

The following XML sample is composed of **`<temp>`** groups. Each **`<temp>`** group contains a **`<month>`** element and a **`<degree>`** element, which contains the average temperature for that month:

```
<temps>
  <temp>
    <month>Jan</month>
    <degree>11</degree>
  </temp>
  <temp>
    <month>Feb</month>
    <degree>14</degree>
  </temp>
  <temp>
    <month>Mar</month>
    <degree>16</degree>
  </temp>
  <temp>
    <month>Apr</month>
    <degree>20</degree>
  </temp>
  <temp>
    <month>May</month>
    <degree>31</degree>
  </temp>
  <temp>
    <month>Jun</month>
    <degree>34</degree>
  </temp>
  <temp>
    <month>Jul</month>
    <degree>39</degree>
  </temp>
  <temp>
    <month>Aug</month>
    <degree>38</degree>
  </temp>
  <temp>
    <month>Sep</month>
    <degree>24</degree>
  </temp>
  <temp>
    <month>Oct</month>
    <degree>28</degree>
  </temp>
  <temp>
    <month>Nov</month>
    <degree>18</degree>
  </temp>
  <temp>
    <month>Dec</month>
    <degree>8</degree>
  </temp>
</temps>
```

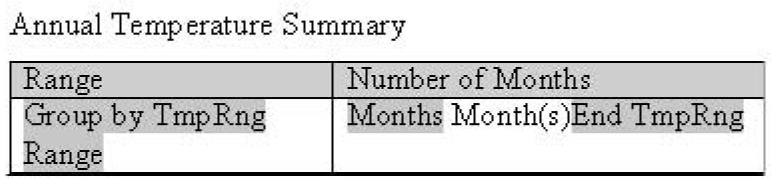
You want to display this data in a format showing temperature ranges and a count of the months that have an average temperature to satisfy those ranges, as follows:


Annual Temperature Summary

| Range | Number of Months |
|---|---|
| 0 F to 10 F | 1 Month(s) |
| 10 F to 20 F | 4 Month(s) |
| 20 F to 30 F | 3 Month(s) |
| 30 F to 40 F | 4 Month(s) |

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

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


Annual Temperature Summary

| Range | Number of Months |
|---|---|
| Group by TmpRng Range | Months Month(s)End TmpRng |

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

| Default Text Entry | Form Field Help Text Entry |
|---|---|
| Group by TmpRng | **`<?for-each-group: temp;floor(degree div 10?>`** **`<?sort:floor(degree div 10)?>`** |
| Range | **`<?concat(floor(degree div 10)*10,' F to ',floor(degree div 10)*10+10, F')?>`** |
| Months | **`<?count(current-group())?>`** |
| End TmpRng | **`<?end for-each-group?>`** |

Note the following about the form field tags:

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

  In this case, it returns the value of the **`<degree>`** element, which is then divided by 10. This will generate the following values from the XML data: 1, 1, 1, 2, 3, 3, 3, 3, 2, 2, 1, and 0.

These are sorted, so that when processed, the following four groups will be created: 0, 1, 2, and 3.

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

  Therefore, for the first group, 0, the row heading displays 0 to (0 +10), or "0 F to 10 F".
- The **`<?count(current-group())?>`** uses the count function to count the members of the current group (the number of temperatures that satisfy the range).
- The **`<?end for-each-group?>`** tag closes out the grouping.

## Chart Support

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

The BI Beans graph DTD is fully documented in the following technical note available from the Oracle Technology Network: "DTD for Customizing Graphs in Oracle Reports."

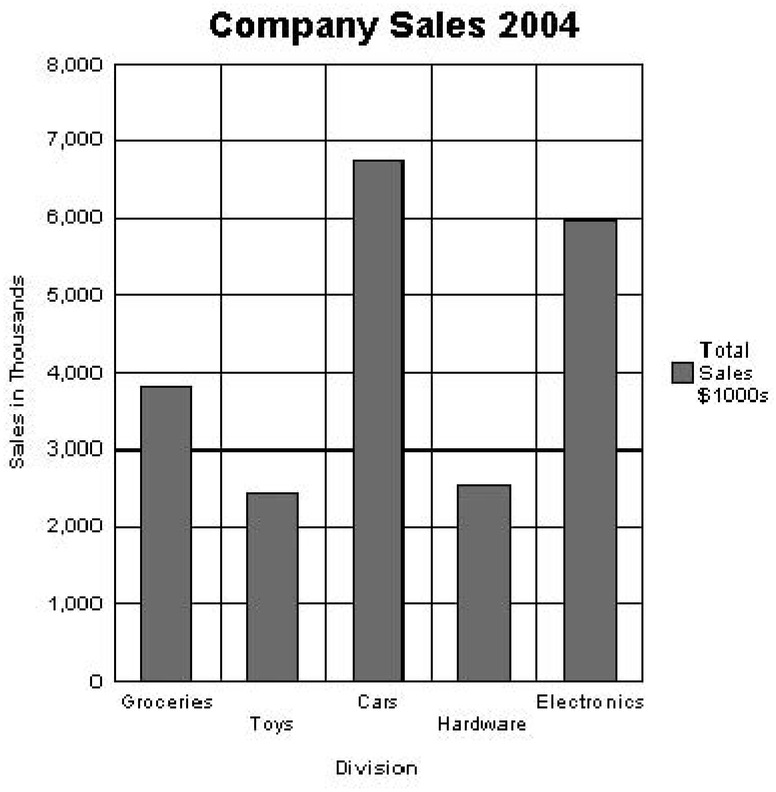
The following summarizes the steps to add a chart to your template. These steps will be discussed in detail in the example that follows::

1. Insert a dummy image in your template to define the size and position of your chart.
2. Add the data definition for the chart to the Alternative text box of the dummy image. The data definition of the chart includes XSL commands ....
3. At runtime XML Publisher calls the BI Beans applications to render the image that is then inserted into the final output document.

### Adding a Sample Chart

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

```
<sales year=2004>
 <division>
  <name>Groceries</name>
  <totalsales>3810</totalsales>
  <costofsales>2100</costofsales>
 </division>
 <division>
  <name>Toys</name>
  <totalsales>2432</totalsales>
  <costofsales>1200</costofsales>
 </division>
 <division>
  <name>Cars</name>
  <totalsales>6753</totalsales>
  <costofsales>4100</costofsales>
 </division>
 <division>
  <name>Hardware</name>
  <totalsales>2543</totalsales>
  <costofsales>1400</costofsales>
 </division>
 <division>
  <name>Electronics</name>
  <totalsales>5965</totalsales>
  <costofsales>3560</costofsales>
 </division>
</sales>
```
This example will show how to insert a chart into your template to display this data as follows:



Note the following attributes of this chart:

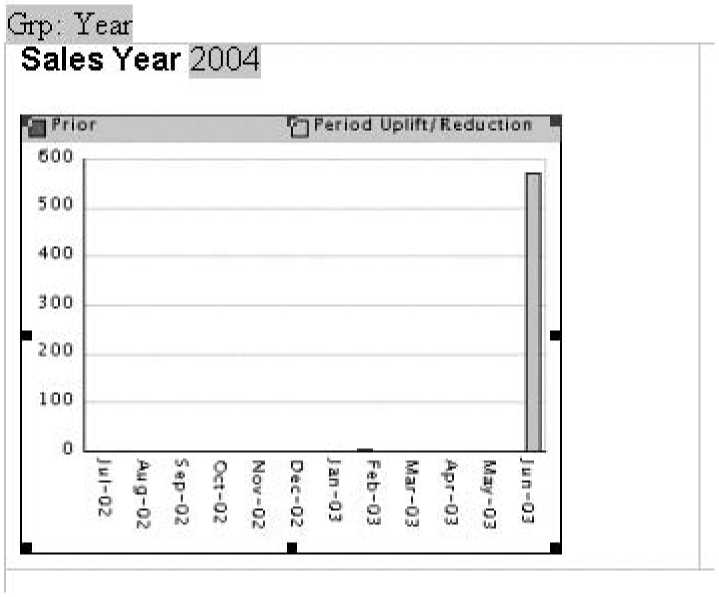
- The style is a vertical bar chart.
- The chart displays a background grid.
- The components are colored.
- Sales totals are shown as Y-axis labels
- Divisions are shown as X-axis labels
- The chart is titled
- The chart displays a legend

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

### Inserting the Dummy Image

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

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

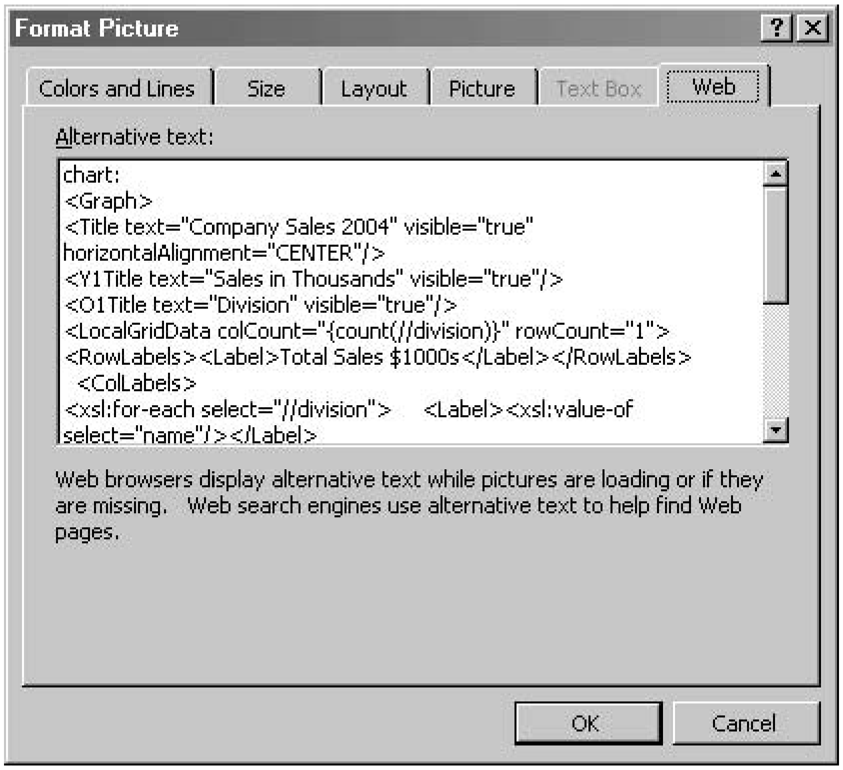


The image can be embedded inside a for-each loop like any other form field if you want the chart to be repeated in the output based on the repeating data. In this example, the chart is defined within the sales year group so that a chart will be generated for each year of data present in the XML file.

Open the **Format Picture** palette for the image that you created and select the **Web** tab. Use the **Alternative text** entry box to enter the code to define the chart characteristics and data definition for the chart.

## Adding Code to the Alternative Text Box

The following graphic shows an example of the XML Publisher code in the **Format Picture Alternative text** box:



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

Web browsers display alternative text while pictures are loading or if they are missing. Web search engines use alternative text to help find Web pages.

The content of the **Alternative text** represents the chart that will be rendered in the final document. For this chart, the text is as follows:

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

The first element of your chart text must be the **`chart:`** element to inform the RTF parser that the following code describes a chart object.

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

The following code defines the chart type and attributes:

```
<Title text="Company Sales 2004" visible="true" horizontalAlignme
nt="CENTER"/>
 <Y1Title text="Sales in Thousands" visible="true"/>
 <O1Title text="Division" visible="true"/>
```

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

```
<Title text="{CHARTTITLE}" visible="true" horizontalAlighment="CE
NTER"/>
```

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

The next section defines the column and row labels:

```
<LocalGridData colCount="{count(//division)}" rowCount="1">
  <RowLabels>
   <Label>Total Sales $1000s</Label>
  </RowLabels>
  <ColLabels>
   <xsl:for-each select="//division">
    <Label>
      <xsl:value-of select="name"/>
    </Label>
   </xsl:for-each>
  </ColLabels>
```

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

`colCount="{count(//division)}"`
The `rowCount` has been hard-coded to 1. This value defines the number of sets of data to be charted. In this case it is 1.

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

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

```
 <ColLabels>
   <xsl:for-each select="//division">
    <Label>
      <xsl:value-of select="name"/>
    </Label>
   </xsl:for-each>
  </ColLabels>
```

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

```
<ColLabels>
  <Label>Groceries</Label>
  <Label>Toys</Label>
  <Label>Cars</Label>
  <Label>Hardware</Label>
  <Label>Electronics</Label>
</ColLabels>
```

The next section defines the actual data values to chart:

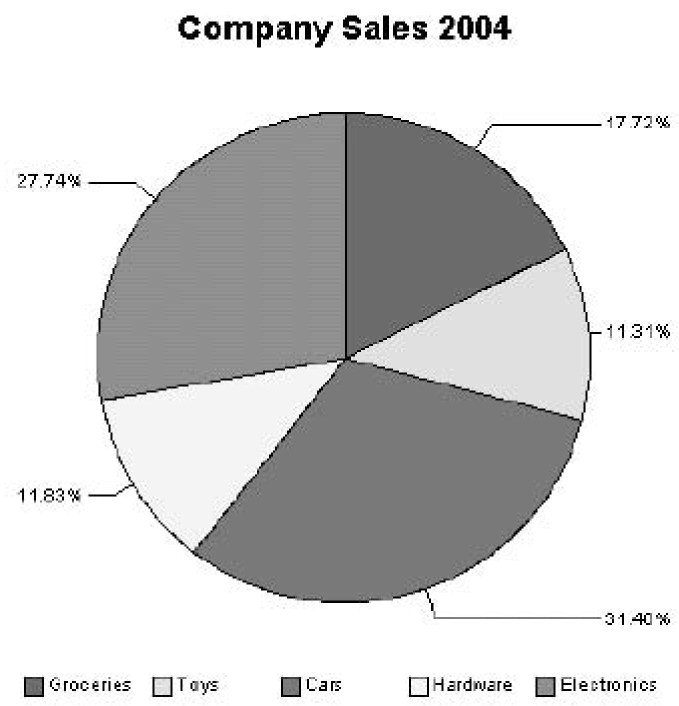
```
<DataValues>
   <RowData>
    <xsl:for-each select="//division">
      <Cell>
       <xsl:value-of select="totalsales"/>
     </Cell>
    </xsl:for-each>
   </RowData>
  </DataValues>
```

Similar to the labels section, the code loops through the data to build the XML that is passed to the BI Beans rendering engine. This will generate the following XML:

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

## Additional Chart Samples

You can also display this data in a pie chart as follows:

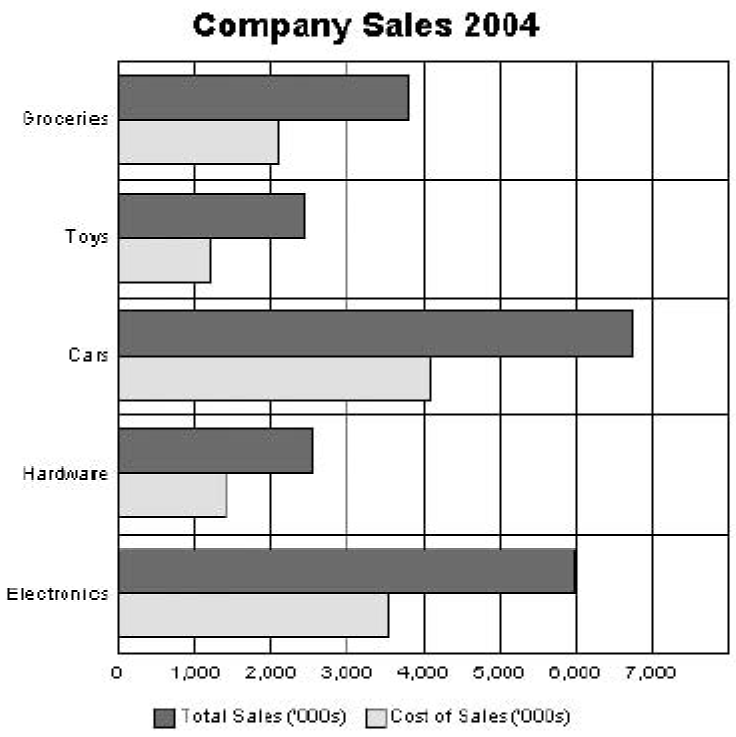


The following is the code added to the template to render this chart at runtime:

```
chart:
<Graph graphType="PIE">
 <Title text="Company Sales 2004" visible="true"
   horizontalAlignment="CENTER"/>
 <LocalGridData rowCount="{count(//division)}" colCount="1">
  <RowLabels>
  <xsl:for-each select="//division">
  <Label>
   <xsl:value-of select="name"/>
  </Label>
  </xsl:for-each>
  </RowLabels>
  <DataValues>
   <xsl:for-each select="//division">
    <RowData>
      <Cell>
       <xsl:value-of select="totalsales"/>
      </Cell>
    </RowData>
   </xsl:for-each>
  </DataValues>
 </LocalGridData>
</Graph>
```

### Horizontal Bar Chart Sample

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



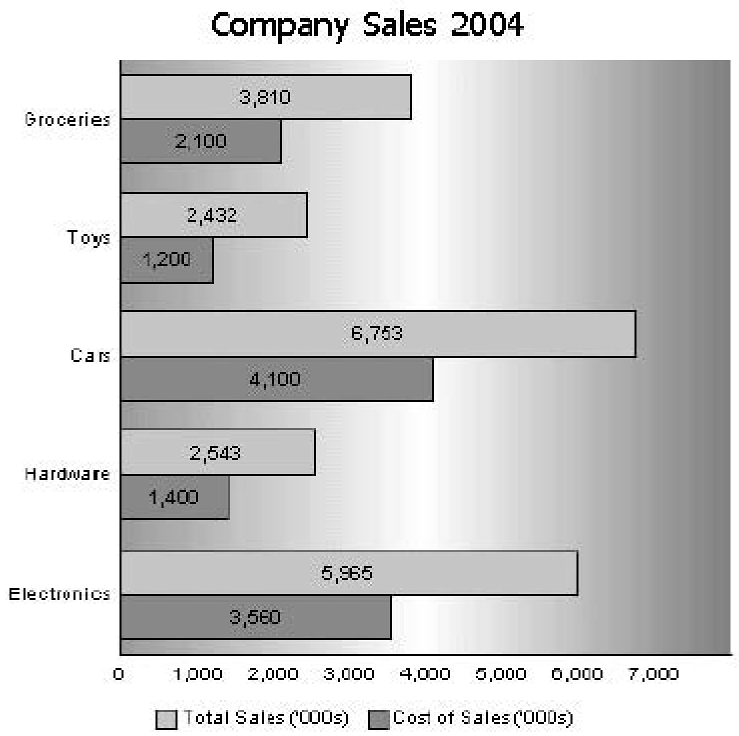
The following code defines this chart in the template:

```
chart:
<Graph graphType = "BAR_HORIZ_CLUST">
 <Title text="Company Sales 2004" visible="true" horizontalAlignme
nt="CENTER"/>
 <LocalGridData colCount="{count(//division)}" rowCount="2">
 <RowLabels>
  <Label>Total Sales ('000s)</Label>
  <Label>Cost of Sales ('000s)</Label>
 </RowLabels>
 <ColLabels>
  <xsl:for-each select="//division">
   <Label><xsl:value-of select="name"/></Label>
  </xsl:for-each>
 </ColLabels>
 <DataValues>
  <RowData>
   <xsl:for-each select="//division">
     <Cell><xsl:value-of select="totalsales"/></Cell>
   </xsl:for-each>
  </RowData>
  <RowData>
   <xsl:for-each select="//division">
    <Cell><xsl:value-of select="costofsales"/></Cell>
    </xsl:for-each>
  </RowData>
  </DataValues>
 </LocalGridData>
</Graph>
```

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

## Changing the Appearance of Your Chart

There are many attributes available from the BI Beans graph DTD that you can manipulate to change the look and feel of your chart. For example the previous chart can be changed to remove the grid, place a graduated background, and change the bar colors and fonts:



The code to support this is as follows:

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

The colors for the bars are defined in the `SeriesIte`m section. The colors are defined in hexadecimal format as follows:

```
<SeriesItems>
  <Series id="0" color="#ffcc00"/>
  <Series id="1" color="#ff6600"/>
</SeriesItems>
```

The following code hides the chart grid:

```
  <O1MajorTick visible="false"/>
  <X1MajorTick visible="false"/>
  <Y1MajorTick visible="false"/>
  <Y2MajorTick visible="false"/>
```

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

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

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

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

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

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

## Advanced Design Options

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

## Using XSL Elements

You can use any XSL element in your template by inserting the XSL syntax into a form field.

If you are using the basic RTF method, you cannot insert XSL syntax directly into your template. XML Publisher has extended the following XSL elements for use in RTF templates.

To use these in a basic-method RTF template, you must use the XML Publisher Tag form of the XSL element. If you are using form fields, use either option.

### Apply a Template Rule

Use this element to apply a template rule to the current element's child nodes.

**XSL Syntax**: `<xsl:apply-templates select="name">`

**XML Publisher Tag**: `<?apply:`*`name`*`?>`

This function applies to `<xsl:template-match="`*`n`*`">` where *n* is the element name.

### Copy the Current Node

Use this element to create a copy of the current node.

**XSL Syntax**: `<xsl:copy-of select="name">`

**XML Publisher Tag**: `<?copy-of:`*`name`*`?>`

### Call Template

Use this element to call a named template to be inserted into or applied to the current template. For example, use this feature to render a table multiple times.

**XSL Syntax**: `<xsl:call-template name="name">`

**XML Publisher Tag**: `<?call-template:`*`name`*`?>`

### Template Declaration

Use this element to apply a set of rules when a specified node is matched.

**XSL Syntax**: `<xsl:template name="name">`

**XML Publisher Tag**: `<?template:`*`name`*`?>`

### Variable Declaration

Use this element to declare a local or global variable.

**XSL Syntax**: `<xsl:variable name="name">`

**XML Publisher Tag**: `<?variable:`*`name`*`?>`

**Example**:

```
<xsl:variable name="color" select="'red'"/>
```

Assigns the value "red" to the "color" variable. The variable can then be referenced in the template.

### Import Stylesheet

Use this element to import the contents of one style sheet into another.

> **Note:** An imported style sheet has lower precedence than the importing style sheet.

**XSL Syntax**: `<xsl:import href="url">`

**XML Publisher Tag**: `<?import:`*`url`*`?>`

### Define the Root Element of the Stylesheet

This and the `<xsl:stylesheet>` element are completely synonymous elements. Both are used to define the root element of the style sheet.

> **Note:** An included style sheet has the same precedence as the including style sheet.

**XSL Syntax**: `<xsl:stylesheet xmlns:x="url">`

**XML Publisher Tag**: `<?namespace:x=url?>`

> **Note:** The namespace must be declared in the template. See Namespace Support, page 2-19.

### Native XSL Number Formatting

The native XSL format-number function takes the basic format:

**`format-number(number,format,[decimalformat])`**

| Parameter | Description |
|---|---|
| number | Required. Specifies the number to be formatted. |
| format | Required. Specifies the format pattern. Use the following characters to specify the pattern:<br>• # (Denotes a digit. Example: ####)<br>• 0 (Denotes leading and following zeros. Example: 0000.00)<br>• · (The position of the decimal point Example: ###.##)<br>• , (The group separator for thousands. Example: ###,###.##)<br>• % (Displays the number as a percentage. Example: ##%)<br>• ; (Pattern separator. The first pattern will be used for positive numbers and the second for negative numbers) |
| decimalformat | Optional. For more information on the decimal format please consult any basic XSLT manual. |

### Using FO Elements

You can use the native FO syntax inside the Microsoft Word form fields.

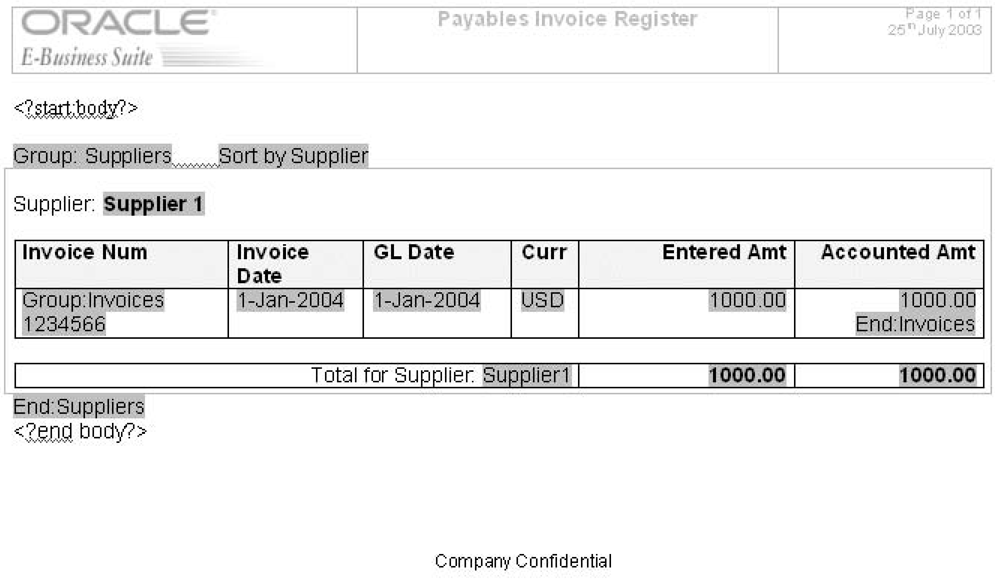
For more information on XSL-FO see the W3C Website at http://www.w3.org/2002/08/XSLFOsummary.html

## Best Practices

### Using Tables

To optimize the exact placement of elements when the template is transformed into XSL, it is recommended that you use tables to define the placement and alignment.

Note the use of tables in the Payables Invoice Register:



| ORACLE E-Business Suite | Payables Invoice Register | Page 1 of 1 25th July 2003 |
|---|---|---|

<?start:body?>

Group: Suppliers Sort by Supplier

Supplier: **Supplier 1**

| Invoice Num | Invoice Date | GL Date | Curr | Entered Amt | Accounted Amt |
|---|---|---|---|---|---|
| Group:Invoices 1234566 | 1-Jan-2004 | 1-Jan-2004 | USD | 1000.00 | 1000.00 End:Invoices |

| Total for Supplier: Supplier1 | **1000.00** | **1000.00** |
|---|---|---|

End:Suppliers
<?end body?>

Company Confidential

A table is used in the header to place the image, the title, and the date in exact positions. By using a table, each element can be aligned within its own cell; thereby allowing a left alignment for the image, a center alignment for the title, and a right alignment for the date and page number.

A table is also used for the totals line of the report to achieve alignment with the entries in the Invoices table.

Tables used for formatting only can be hidden at runtime by turning off (hiding) the table gridlines.

# 3

# Creating a PDF Template

This chapter covers the following topics:

- PDF Template Overview
- Designing the Layout
- Adding Markup to the Template Layout
- Adding Page Numbers and Page Breaks
- Performing Calculations
- Completed PDF Template
- Runtime Behavior
- Creating a Template from a Downloaded PDF

## PDF Template Overview

To create a PDF template, you take any existing PDF document and apply the XML Publisher markup. Because the source of the PDF document does not matter, you have multiple design options. For example:

- Design the layout of your template using any application that generates documents that can be converted to PDF
- Scan a paper document to use as a template
- Download a PDF document from a third party Website

> **Note:** The steps required to create a template from a third-party PDF depend on whether form fields have been added to the document. For more information, see Creating a Template from a Downloaded PDF, page 3-14.

If you are designing the layout, note that once you have converted to PDF, your layout is treated like a set background. When you mark up the template, you draw fields on top of this background. To edit the layout, you must edit your original document and then convert back to PDF.

For this reason, the PDF template is not recommended for documents that will require frequent updates to the layout. However, it is appropriate for forms that will have a fixed layout, such as invoices or purchase orders.

### Supported Modes

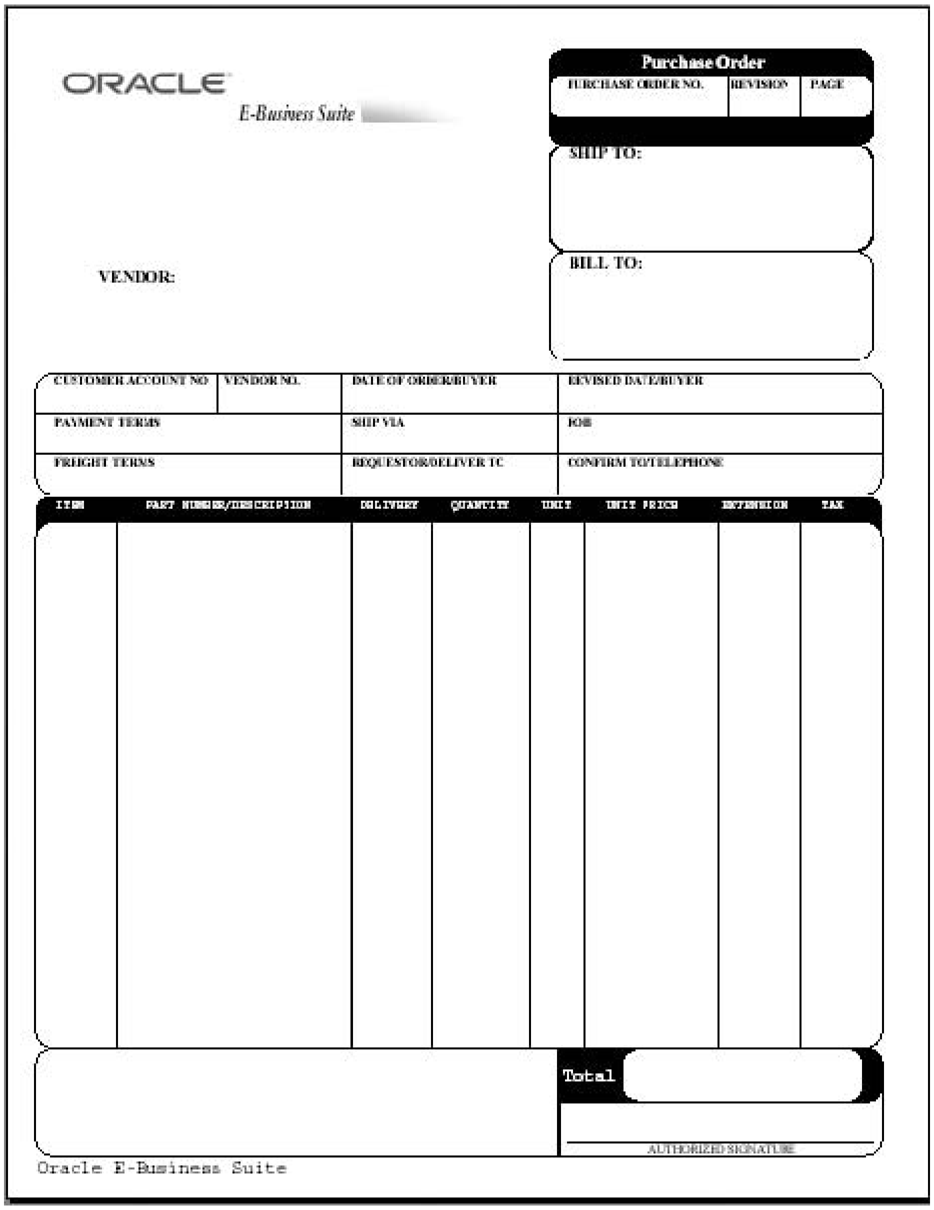
XML Publisher supports Adobe Acrobat 5.0 (PDF specification version 1.4). If you are using Adobe Acrobat 6.0, use the **Reduce File Size Option** (from the **File** menu) to save your file as Adobe Acrobat 5.0 compatible.

For PDF conversion, XML Publisher supports any PDF conversion utility, such as Adobe Acrobat Distiller.

## Designing the Layout

To design the layout of your template you can use any desktop application that generates documents that can be converted to PDF. Or, scan in an original paper document to use as the background for the template.

The following is the layout for a sample purchase order. It was designed using Microsoft Word and converted to PDF using Adobe Acrobat Distiller.



ORACLE

E-Business Suite

**Purchase Order**

| PURCHASE ORDER NO. | REVISION | PAGE |
|---|---|---|
| | | |

SHIP TO:

BILL TO:

VENDOR:

| | | | |
|---|---|---|---|
| CUSTOMER ACCOUNT NO / VENDOR NO. | | DATE OF ORDER/BUYER | REVISED DATE/BUYER |
| PAYMENT TERMS | | SHIP VIA | FOB |
| FREIGHT TERMS | | REQUESTOR/DELIVER TO | CONFIRM TO/TELEPHONE |

| ITEM | PART NUMBER/DESCRIPTION | DELIVERY | QUANTITY | UNIT | UNIT PRICE | EXTENSION | TAX |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

Total

AUTHORIZED SIGNATURE

Oracle E-Business Suite

The following is the XML data that will be used as input to this template:

```
<?xml version="1.0"?>
<POXPRPOP2>
 <G_HEADERS>
  <POH_PO_NUM>1190-1</POH_PO_NUM>
  <POH_REVISION_NUM>0</POH_REVISION_NUM>
  <POH_SHIP_ADDRESS_LINE1>3455 108th Avenue</POH_SHIP_ADDRESS_LINE
1>
<POH_SHIP_ADDRESS_LINE2></POH_SHIP_ADDRESS_LINE2>
<POH_SHIP_ADDRESS_LINE3></POH_SHIP_ADDRESS_LINE3>
<POH_SHIP_ADR_INFO>Seattle, WA 98101</POH_SHIP_ADR_INFO>
<POH_SHIP_COUNTRY>United States</POH_SHIP_COUNTRY>
<POH_VENDOR_NAME>Allied Manufacturing</POH_VENDOR_NAME>
<POH_VENDOR_ADDRESS_LINE1>1145 Brokaw Road</POH_VENDOR_ADDRESS_LIN
E1>
<POH_VENDOR_ADR_INFO>San Jose, CA 95034</POH_VENDOR_ADR_INFO>
<POH_VENDOR_COUNTRY>United States</POH_VENDOR_COUNTRY>
<POH_BILL_ADDRESS_LINE1>90 Fifth Avenue</POH_BILL_ADDRESS_LINE1>
<POH_BILL_ADR_INFO>New York, NY 10022-3422</POH_BILL_ADR_INFO>
<POH_BILL_COUNTRY>United States</POH_BILL_COUNTRY>
<POH_BUYER>Smith, J</POH_BUYER>
<POH_PAYMENT_TERMS>45 Net (terms date + 45)</POH_PAYMENT_TERMS>
<POH_SHIP_VIA>UPS</POH_SHIP_VIA>
<POH_FREIGHT_TERMS>Due</POH_FREIGHT_TERMS>
<POH_CURRENCY_CODE>USD</POH_CURRENCY_CODE>
<POH_CURRENCY_CONVERSION_RATE></POH_CURRENCY_CONVERSION_RATE>
<LIST_G_LINES>
<G_LINES>
<POL_LINE_NUM>1</POL_LINE_NUM>
<POL_VENDOR_PRODUCT_NUM></POL_VENDOR_PRODUCT_NUM>
<POL_ITEM_DESCRIPTION>PCMCIA II Card Holder</POL_ITEM_DESCRIPTION>
<POL_QUANTITY_TO_PRINT></POL_QUANTITY_TO_PRINT>
<POL_UNIT_OF_MEASURE>Each</POL_UNIT_OF_MEASURE>
<POL_PRICE_TO_PRINT>15</POL_PRICE_TO_PRINT>
<C_FLEX_ITEM>CM16374</C_FLEX_ITEM>
<C_FLEX_ITEM_DISP>CM16374</C_FLEX_ITEM_DISP>
<PLL_QUANTITY_ORDERED>7500</PLL_QUANTITY_ORDERED>
<C_AMOUNT_PLL>112500</C_AMOUNT_PLL>
<C_AMOUNT_PLL_DISP> 112,500.00 </C_AMOUNT_PLL_DISP>
</G_LINES>
</LIST_G_LINES>
<C_AMT_POL_RELEASE_TOTAL_ROUND>312420/<C_AMT_POL_RELEASE_TOTAL_ROU
ND>
</G_HEADERS>
</POXPRPOP2>
```

## Adding Markup to the Template Layout

After you have converted your document to PDF, you define form fields that will display the data from the XML input file. These form fields are placeholders for the data.

The process of associating the XML data to the PDF template is the same as the process for the RTF template. See: Associating the XML Data to the Template Layout: Associating the XML data to the template layout, page 2- 2

When you draw the form fields in Adobe Acrobat, you are drawing them *on top* of the layout that you designed. There is not a relationship between the design elements on your template and the form fields. You therefore must place the fields exactly where you want the data to display on the template.

## Creating a Placeholder

You can define a placeholder as text, a check box, or a radio button, depending on how you want the data presented.

### Naming the Placeholder

When you enter a name for the placeholder, enter either the XML source field name or assign a different, unique name.

**Note:** The placeholder name must not contain the "." character.

If you assign a different name, you must map the template field to the data source field when you register the template in the Template Manager. Mapping requires that you load the XML schema. If you give the template field the same name as the XML source field, no mapping is required.

For information on mapping fields in the Template Manager, see Mapping PDF Template Fields, page 5- 7 .

### Creating a Text Placeholder

To create a text placeholder in your PDF document:

1. Select the **Form Tool** from the Acrobat toolbar.
2. Draw a form field box in the position on the template where you want the field to display. Drawing the field opens the **Field Properties** dialog box.
3. In the **Name** field of the **Field Properties** dialog box, enter a name for the field.
4. Select **Text** from the **Type** drop down menu.

   You can use the **Field Properties** dialog box to set other attributes for the placeholder. For example, enforce maximum character size, set field data type, data type validation, visibility, formatting
5. If the field is not placed exactly where desired, drag the field for exact placement.

### Supported Field Properties Options

XML Publisher supports the following options available from the **Field Properties** dialog box. For more information about these options, see the Adobe Acrobat documentation.

- **Appearance**
  - Border Settings: color, background, width, and style
  - Text Settings: color, font, size
  - Common Properties: read only, required, visible/hidden, orientation (in degrees)
  - Border Style
- **Options** tab
  - Multi-line
  - Scrolling Text

- **Format** tab - Number category options only
- **Calculate** tab - all calculation functions

### Creating a Check Box

A check box is used to present options from which more than one can be selected. Each check box represents a different data element. You define the value that will cause the check box to display as "checked."

For example, a form contains a check box listing of automobile options such as Power Steering, Power Windows, Sunroof, and Alloy Wheels. Each of these represents a different element from the XML file. If the XML file contains a value of "Y" for any of these fields, you want the check box to display as checked. All or none of these options may be selected.

To create a check box field:

1. Insert the form field.
2. In the **Field Properties** dialog box, enter a **Name** for the field.
3. Select **Check Box** from the **Type** drop down list.
4. Select the **Options** tab.
5. In the **Export Value** field enter the value that the XML data field should match to enable the "checked" state.

   For the example, enter "Y" for each check box field.

#### Creating a Radio Button Group

A radio button group is used to display options from which only one can be selected.

For example, your XML data file contains a field called <SHIPMENT_METHOD>. The possible values for this field are "Standard" or "Overnight". You represent this field in your form with two radio buttons, one labeled "Standard" and one labeled "Overnight". Define both radio button fields as placeholders for the <SHIPMENT_METHOD> data field. For one field, define the "on" state when the value is "Standard". For the other, define the "on" state when the value is "Overnight".

To create a radio button group:

1. Insert the form field.
2. On the **Field Properties** dialog box, enter a **Name** for the field. Each radio button you define to represent this value can be named differently, but must be mapped to the same XML data field.
3. Select **Radio Button** from the **Type** drop down list.
4. Select the **Options** tab.
5. In the **Export Value** field enter the value that the XML data field should match to enable the "on" state.

   For the example, enter "Standard" for the field labeled "Standard". Enter "Overnight" for the field labeled "Overnight".

## Defining Groups of Repeating Fields

In the PDF template, you explicitly define the area on the page that will contain the repeating fields. For example, on the purchase order template, the repeating fields should display in the block of space between the Item header row and the Total field.

**To define the area to contain the group of repeating fields:**

1. Insert a form field at the beginning of the area that is to contain the group.
2. In the **Name** field of the **Field Properties** window, enter any name you choose. This field is not mapped.
3. Select **Text** from the **Type** drop down list.
4. In the **Short Description** field of the **Field Properties** window, enter the following syntax:

   ```
   <?rep_field="BODY_START"?>
   ```

5. Define the end of the group area by inserting a form field at the end of the area the that is to contain the group.
6. In the **Name** field of the **Field Properties** window, enter any name you choose. This field is not mapped.
7. Select **Text** from the **Type** drop down list.
8. In the **Short Description** field of the **Field Properties** window, enter the following syntax:

   ```
   <?rep_field="BODY_END"?>
   ```

**To define a group of repeating fields:**

1. Insert a placeholder for the first element of the group.

   **Note:** The placement of this field in relationship to the BODY_START tag defines the distance between the repeating rows for each occurrence. See Placement of Repeating Fields, page 3-13.

2. For each element in the group, enter the following syntax in the **Short Description** field:

   <?rep_field="T1_G*n*"?>

   where n is the row number of the item on the template.

   For example, the group in the sample report is laid out in three rows.

   - For the fields belonging to the row that begins with "PO_LINE_NUM" enter

     ```
     <?rep_field="T1_G1"?>
     ```

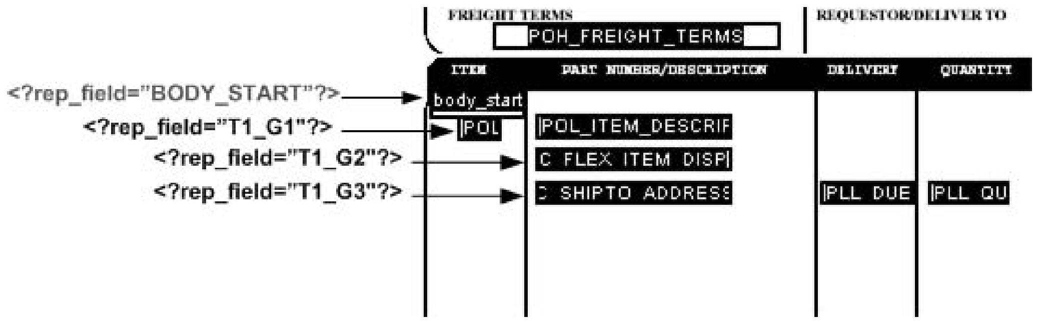
   - For the fields belonging to the row that begins with "C_FLEX_ITEM_DISP" enter

     ```
     <?rep_field="T1_G2"?>
     ```

   - For the fields belonging to the row that begins with "C_SHIP_TO_ADDRESS" enter

     ```
     <?rep_field="T1_G3"?>
     ```

   The following graphic shows the entries for the **Short Description** field:



3. (Optional) Align your fields. To ensure proper alignment of a row of fields, it is recommended that you use Adobe Acrobat's alignment feature.

## Adding Page Numbers and Page Breaks

This section describes how to add the following page-features to your PDF template:

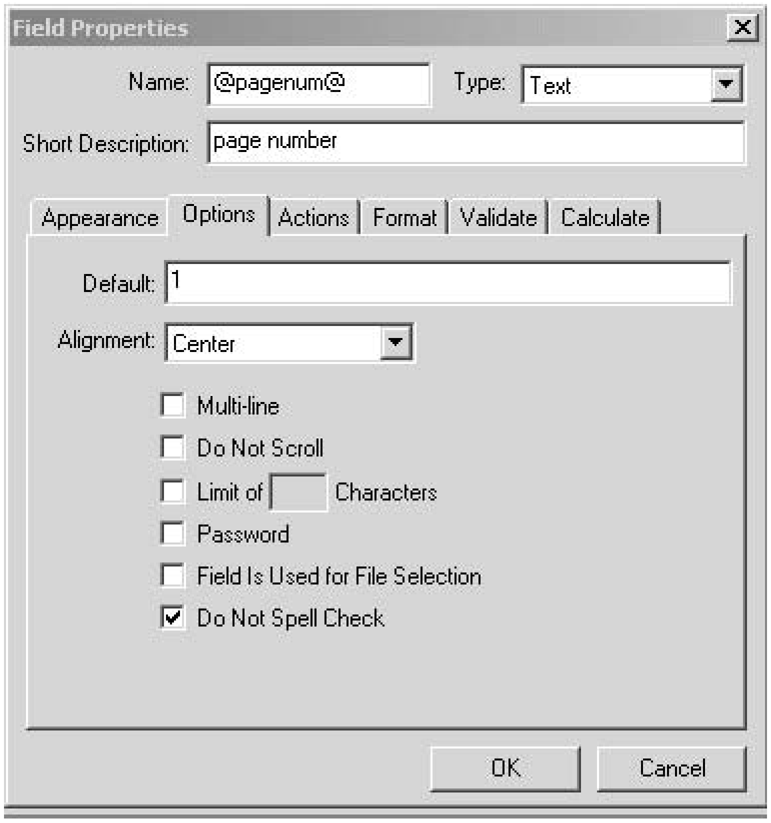
- Page Numbers
- Page Breaks

### Adding Page Numbers

To add page numbers, define a field in the template where you want the page number to appear and enter an initial value in that field as follows:

1. Decide the position on the template where you want the page number to be displayed.
2. Create a placeholder field called **`@pagenum@`** (see Creating a Text Placeholder, page 3- 5 ).
3. Enter a starting value for the page number in the **Default** field. If the XML data includes a value for this field, the start value assigned in the template will be overriden. If no start value is assigned, it will default to 1.

The figure below shows the Field Properties dialog for a page number field:



Name: @pagenum@
Type: Text
Short Description: page number
Default: 1
Alignment: Center
- [ ] Multi-line
- [ ] Do Not Scroll
- [ ] Limit of Characters
- [ ] Password
- [ ] Field Is Used for File Selection
- [✓] Do Not Spell Check

## Adding Page Breaks

You can define a page break in your template to occur after a repeatable field. To insert a page break after the occurrence of a specific field, add the following to the syntax in the **Short Description** field of the Field Properties dialog box:

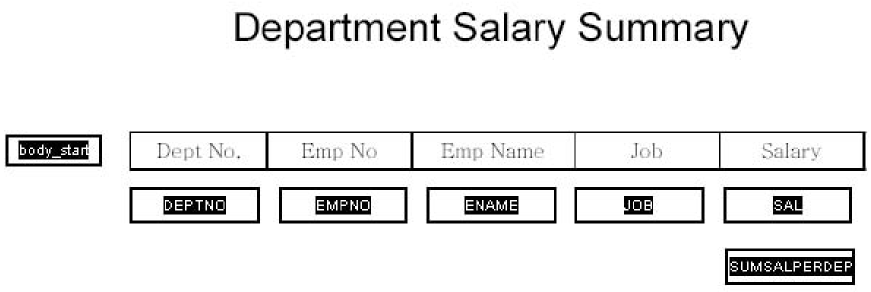
```
page_break="yes"
```

For example:

```
<?rep_field="T1_G3", page_break="yes"?>
```

The following example demonstrates inserting a page break in a template. The XML sample contains salaries of employees by department:

```
<?xml version="1.0"?>
<! -  Generated by Oracle Reports version 6.0.8.22.0  - >
<ROOT>
  <LIST_G_DEPTNO>
    <G_DEPTNO>
      <DEPTNO>10</DEPTNO>
      <LIST_G_EMPNO>
        <G_EMPNO>
          <EMPNO>7782</EMPNO>
          <ENAME>CLARK</ENAME>
          <JOB>MANAGER</JOB>
          <SAL>2450</SAL>
        </G_EMPNO>
        <G_EMPNO>
          <EMPNO>7839</EMPNO>
          <ENAME>KING</ENAME>
          <JOB>PRESIDENT</JOB>
          <SAL>5000</SAL>
        </G_EMPNO>
        <G_EMPNO>
          <EMPNO>125</EMPNO>
          <ENAME>KANG</ENAME>
          <JOB>CLERK</JOB>
          <SAL>2000</SAL>
        </G_EMPNO>
        <G_EMPNO>
          <EMPNO>7934</EMPNO>
          <ENAME>MILLER</ENAME>
          <JOB>CLERK</JOB>
          <SAL>1300</SAL>
        </G_EMPNO>
        <G_EMPNO>
          <EMPNO>123</EMPNO>
          <ENAME>MARY</ENAME>
          <JOB>CLERK</JOB>
          <SAL>400</SAL>
        </G_EMPNO>
        <G_EMPNO>
          <EMPNO>124</EMPNO>
          <ENAME>TOM</ENAME>
          <JOB>CLERK</JOB>
          <SAL>3000</SAL>
        </G_EMPNO>
      </LIST_G_EMPNO>
      <SUMSALPERDEPTNO>9150</SUMSALPERDEPTNO>
    </G_DEPTNO>

    <G_DEPTNO>
      <DEPTNO>30</DEPTNO>
      <LIST_G_EMPNO>
        .
        .
        .

      </LIST_G_EMPNO>
      <SUMSALPERDEPTNO>9400</SUMSALPERDEPTNO>
    </G_DEPTNO>
  </LIST_G_DEPTNO>
  <SUMSALPERREPORT>29425</SUMSALPERREPORT>
</ROOT>
```
We want to report the salary information for each employee by department as shown in the following template:

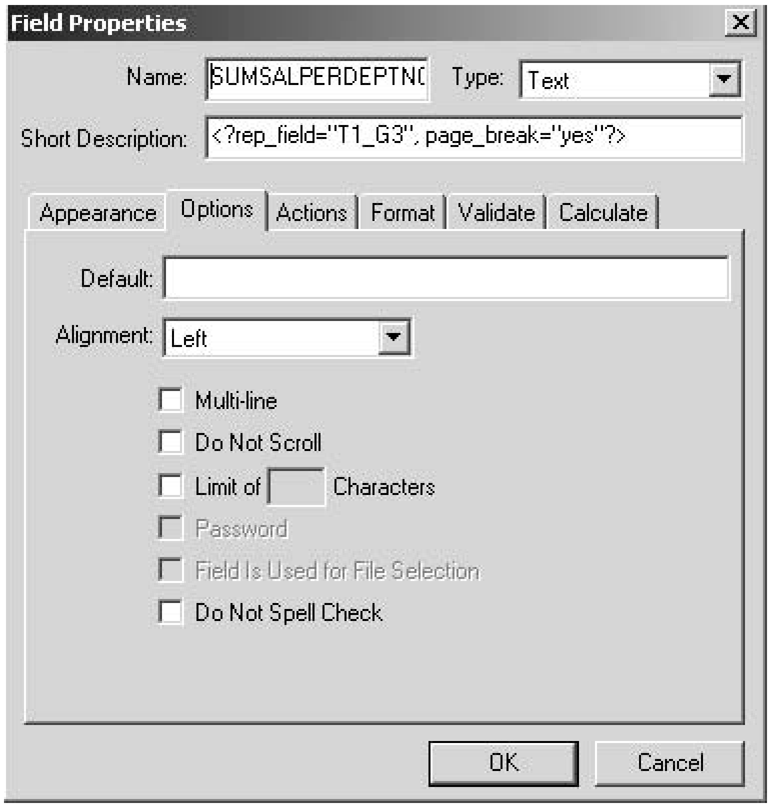

Department Salary Summary

| body_start | Dept No. | Emp No | Emp Name | Job | Salary |
|---|---|---|---|---|---|
| | DEPTNO | EMPNO | ENAME | JOB | SAL |
| | | | | | SUMSALPERDEP |

To insert a page break after each department, insert the page break syntax in the short description for the SUMSALPERDEPTNO field as follows:

```
<?rep_field="T1_G3", page_break="yes"?>
```

The Field Properties dialog box for the field is shown in the following figure:


Field Properties

Name: SUMSALPERDEPTNO Type: Text

Short Description: <?rep_field="T1_G3", page_break="yes"?>

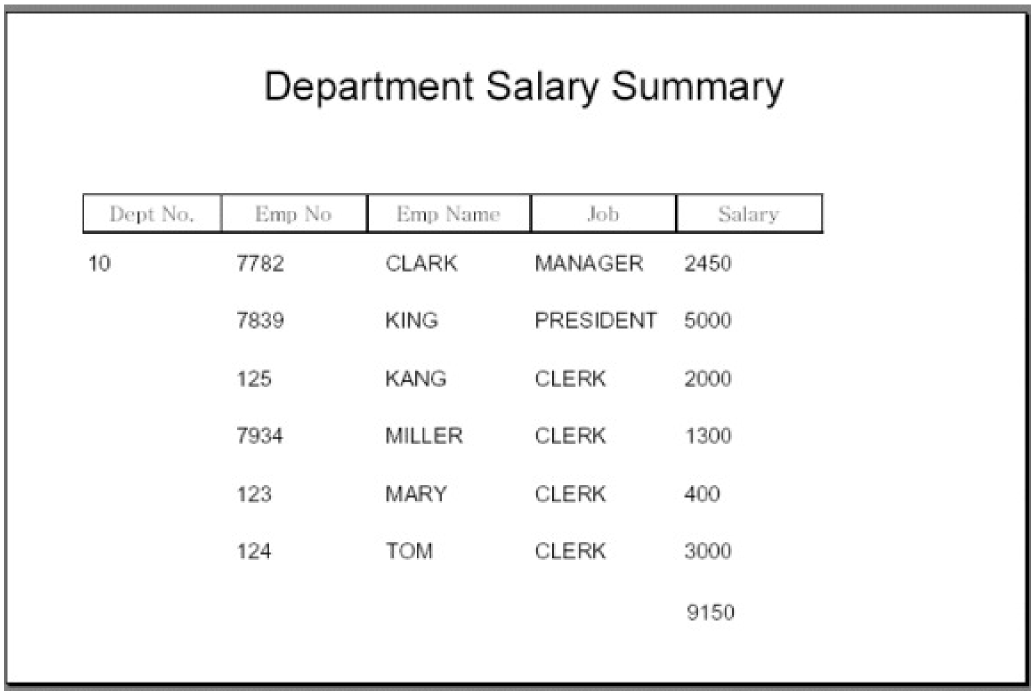
Default:

Alignment: Left

- [ ] Multi-line
- [ ] Do Not Scroll
- [ ] Limit of Characters
- [ ] Password
- [ ] Field Is Used for File Selection
- [ ] Do Not Spell Check

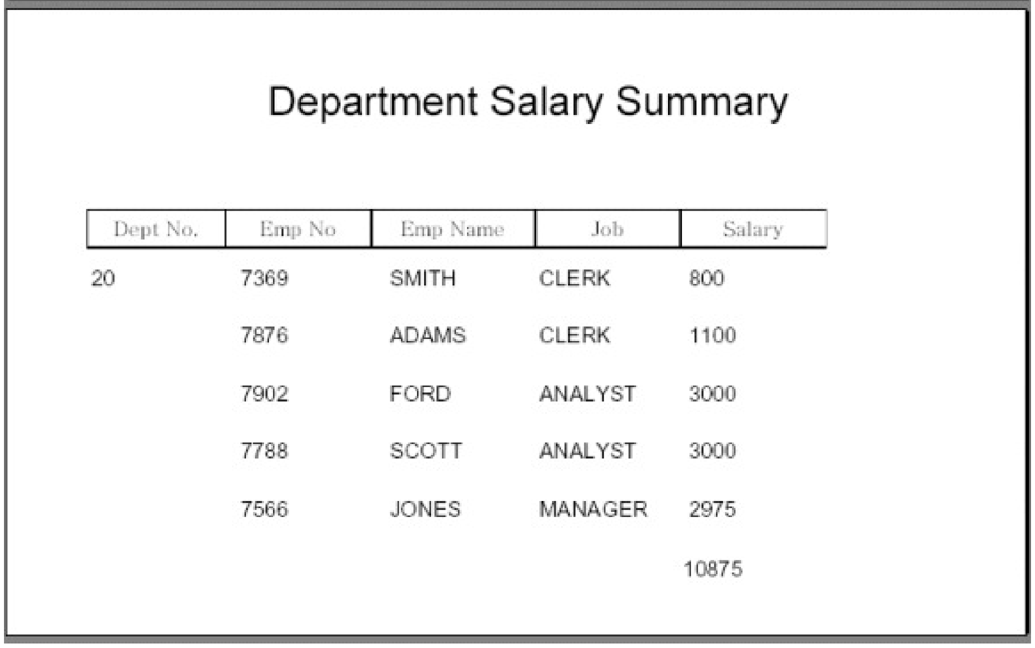
Note that in order for the break to occur, the field must be populated with data from the XML file.

The sample report with data is shown in the following figure:



## Department Salary Summary

| Dept No. | Emp No | Emp Name | Job | Salary |
|---|---|---|---|---|
| 10 | 7782 | CLARK | MANAGER | 2450 |
| | 7839 | KING | PRESIDENT | 5000 |
| | 125 | KANG | CLERK | 2000 |
| | 7934 | MILLER | CLERK | 1300 |
| | 123 | MARY | CLERK | 400 |
| | 124 | TOM | CLERK | 3000 |
| | | | | 9150 |



## Department Salary Summary

| Dept No. | Emp No | Emp Name | Job | Salary |
|---|---|---|---|---|
| 20 | 7369 | SMITH | CLERK | 800 |
| | 7876 | ADAMS | CLERK | 1100 |
| | 7902 | FORD | ANALYST | 3000 |
| | 7788 | SCOTT | ANALYST | 3000 |
| | 7566 | JONES | MANAGER | 2975 |
| | | | | 10875 |

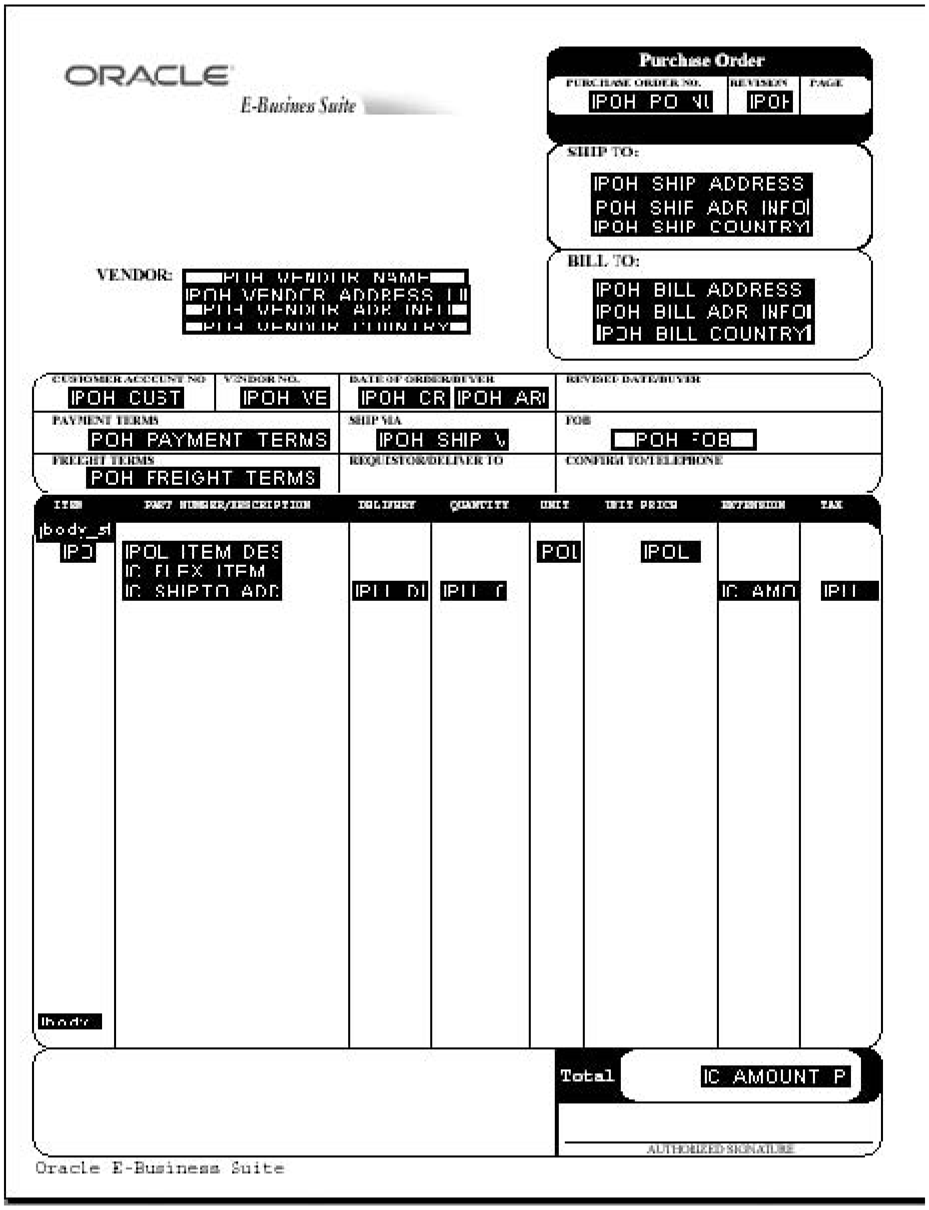
## Performing Calculations

Adobe Acrobat provides a calculation function in the **Field Properties** dialog box. To create a field to display a calculated total on your report:

1. Create a text field to display the calculated total. Give the field any **Name** you choose.
2. In the **Field Properties** dialog box, select the **Format** tab.
3. Select **Number** from the **Category** list.
4. Select the **Calculate** tab.

5. Select the radio button next to "Value is the *operation* of the following fields:"
6. Select **sum** from the drop down list.
7. Select the **Pick...** button and select the fields that you want totaled.

## Completed PDF Template



ORACLE E-Business Suite

Purchase Order

| PURCHASE ORDER NO. | REVISION | PAGE |
|---|---|---|
| POH PO NU | POH | |

SHIP TO:
POH SHIP ADDRESS
POH SHIP ADR INFO
POH SHIP COUNTRY

VENDOR:
POH VENDOR NAME
POH VENDOR ADDRESS LI
POH VENDOR ADR INFO
POH VENDOR COUNTRY

BILL TO:
POH BILL ADDRESS
POH BILL ADR INFO
POH BILL COUNTRY

| | | | |
|---|---|---|---|
| CUSTOMER ACCOUNT NO POH CUST | VENDOR NO. POH VE | DATE OF ORDER/BUYER POH CR POH AR | REVISED DATE/BUYER |
| PAYMENT TERMS POH PAYMENT TERMS | | SHIP VIA POH SHIP V | FOB POH FOB |
| FREIGHT TERMS POH FREIGHT TERMS | | REQUISTOR/DELIVER TO | CONFIRM TO/TELEPHONE |

| ITEM | PART NUMBER/DESCRIPTION | DELIVERY | QUANTITY | UNIT | UNIT PRICE | EXTENSION | TAX |
|---|---|---|---|---|---|---|---|
| body_st POL | POL ITEM DES C FLEX ITEM C SHIPTO ADD | PLL DI | PLL C | POL | POL | C AMO | PLL |
| body | | | | | | | |

Total C AMOUNT P

AUTHORIZED SIGNATURE

Oracle E-Business Suite

## Runtime Behavior

### Placement of Repeating Fields

As already noted, the placement, spacing, and alignment of fields that you create on the template are independent of the underlying form layout. At runtime, XML Publisher places each repeating row of data according to calculations performed on the placement of the rows of fields that you created, as follows:

**First occurrence:**

The first row of repeating fields will display exactly where you have placed them on the template.

**Second occurrence, single row:**

To place the second occurrence of the group, XML Publisher calculates the distance between the BODY_START tag and the first field of the first occurrence. The first field of the second occurrence of the group will be placed this calculated distance below the first occurrence.

**Second occurrence, multiple rows:**

If the first group contains multiple rows, the second occurrence of the group will be placed the calculated distance below the last row of the first occurrence.

The distance between the rows within the group will be maintained as defined in the first occurrence.

### Overflow Data

When multiple pages are required to accommodate the occurrences of repeating rows of data, each page will display identically except for the defined repeating area, which will display the continuation of the repeating data. For example, if the item rows of the purchase order extend past the area defined on the template, succeeding pages will display all data from the purchase order form with the continuation of the item rows.

## Creating a Template from a Downloaded PDF

The steps for creating a template from a downloaded PDF are:

1. Register the Applications data source in the Template Manager.
2. Register the PDF form as a Template in the Template Manager.
3. Use the mapping feature to map the fields from the downloaded PDF form to your data source.

PDF forms downloaded from third party sources may or may not contain the form fields already defined. To determine if the form fields are defined, open the document in Adobe Acrobat and select the **Form Tool**. If the form fields are defined, they will display in the document.

If the form fields are not defined, you must mark up the template. See Mark up the Layout, page 3- 4 for instructions on inserting placeholders and defining groups of repeating fields.

If the form fields are defined, you are ready to upload the document to the Template Manager for field mapping.

# 4 eText Templates

This chapter covers the following topics:

- Introduction
- Structure of eText Templates
- Constructing the Data Tables
- Setup Command Tables
- Expressions, Control Structure, and Functions
- Identifiers, Operators, and Literals

## Introduction

An eText template is an RTF-based template that is used to generate text output for Electronic Funds Transfer (EFT) and Electronic Data Interchange (EDI). At runtime, XML Publisher applies this template to an input XML data file to create an output text file that can be transmitted to a bank or other customer. Because the output is intended for electronic communication, the eText templates must follow very specific format instructions for exact placement of data.

> **Note:** An EFT is an electronic transmission of financial data and payments to banks in a specific fixed-position format flat file (text).
>
> EDI is similar to EFT except it is not only limited to the transmission of payment information to banks. It is often used as a method of exchanging business documents, such as purchase orders and invoices, between companies. EDI data is delimiter-based, and also transmitted as a flat file (text).

Files in these formats are transmitted as flat files, rather than printed on paper. The length of a record is often several hundred characters and therefore difficult to layout on standard size paper.

To accommodate the record length, the EFT and EDI templates are designed using tables. Each record is represented by a table. Each row in a table corresponds to a field in a record. The columns of the table specify the position, length, and value of the field.

These formats can also require special handling of the data from the input XML file. This special handling can be on a global level (for example, character replacement and sequencing) or on a record level (for example, sorting). Commands to perform these functions are declared in command rows. Global level commands are declared in setup tables.

At runtime, XML Publisher constructs the output file according to the setup commands and layout specifications in the tables.

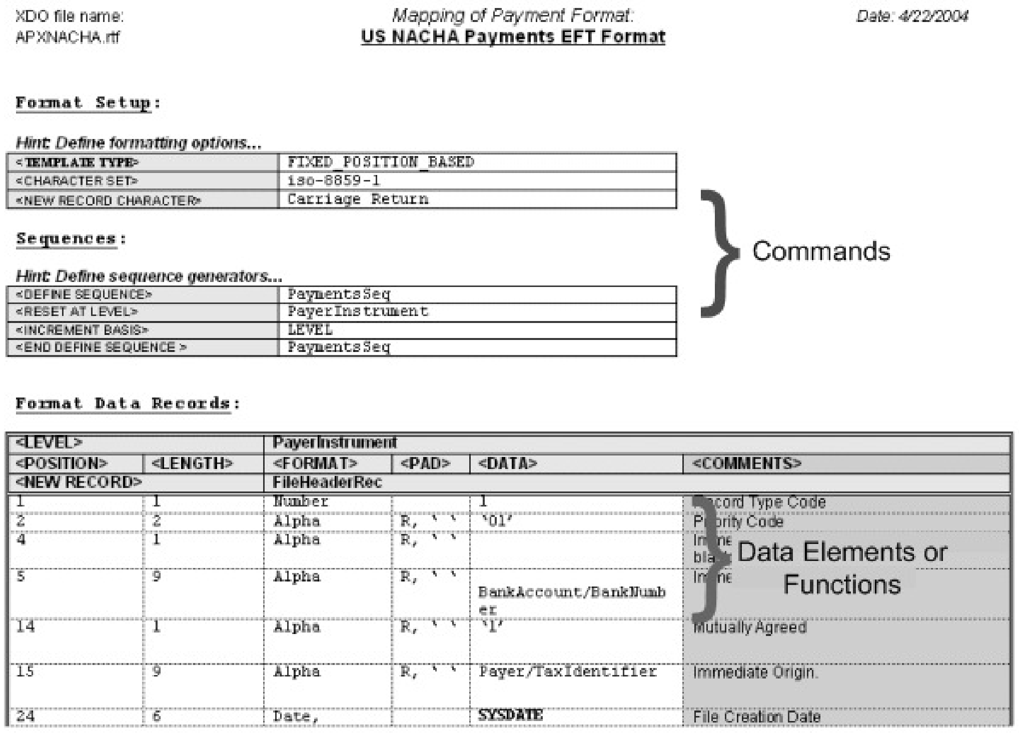
### Prerequisites

This section is intended for users who are familiar with EDI and EFT transactions audience for this section preparers of eText templates will require both functional and technical knowledge. That is, functional expertise to understand bank and country specific payment format requirements and sufficient technical expertise to understand XML data structure and eText specific coding syntax commands, functions, and operations.

## Structure of eText Templates

There are two types of eText templates: fixed-position based (EFT templates) and delimiter-based (EDI templates). The templates are composed of a series of tables. The tables define layout and setup commands and data field definitions. The required data description columns for the two types of templates vary, but the commands and functions available are the same. A table can contain just commands, or it can contain commands and data fields.

The following graphic shows a sample from an EFT template to display the general structure of command and data rows:



XDO file name:
APXNACHA.rtf

*Mapping of Payment Format:*
**US NACHA Payments EFT Format**

*Date: 4/22/2004*

**Format Setup:**

*Hint: Define formatting options...*

| | |
|---|---|
| <TEMPLATE TYPE> | FIXED_POSITION_BASED |
| <CHARACTER SET> | iso-8859-1 |
| <NEW RECORD CHARACTER> | Carriage Return |

**Sequences:**

*Hint: Define sequence generators...*

| | |
|---|---|
| <DEFINE SEQUENCE> | PaymentsSeq |
| <RESET AT LEVEL> | PayerInstrument |
| <INCREMENT BASIS> | LEVEL |
| <END DEFINE SEQUENCE > | PaymentsSeq |

**Format Data Records:**

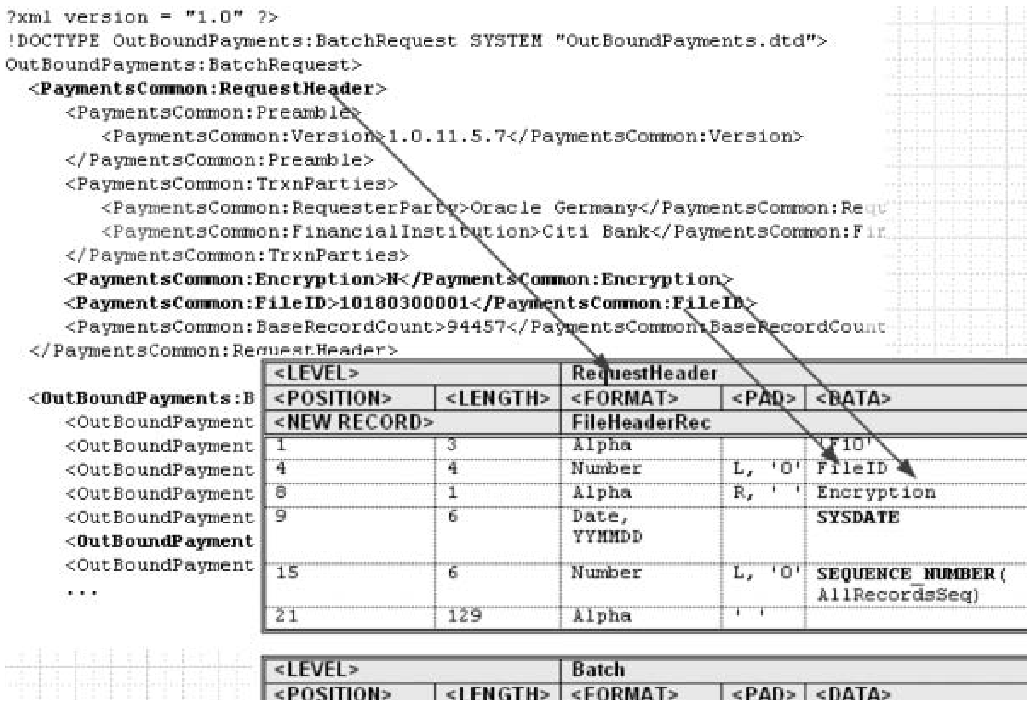
| <LEVEL> | | PayerInstrument | | | |
|---|---|---|---|---|---|
| <POSITION> | <LENGTH> | <FORMAT> | <PAD> | <DATA> | <COMMENTS> |
| <NEW RECORD> | | FileHeaderRec | | | |
| 1 | 1 | Number | | 1 | ecord Type Code |
| 2 | 2 | Alpha | R, ' ' | '01' | P ority Code |
| 4 | 1 | Alpha | R, ' ' | | Im ne bla |
| 5 | 9 | Alpha | R, ' ' | BankAccount/BankNumber | Im ne |
| 14 | 1 | Alpha | R, ' ' | '1' | Mutually Agreed |
| 15 | 9 | Alpha | R, ' ' | Payer/TaxIdentifier | Immediate Origin. |
| 24 | 6 | Date, | | **SYSDATE** | File Creation Date |

Commands that apply globally, or commands that define program elements for the template, are "setup" commands. These must be specified in the initial table(s) of the template. Examples of setup commands are Template Type and Character Set.

In the data tables you provide the source XML data element name (or static data) and the specific placement and formatting definitions required by the receiving bank or entity. You can also define functions to be performed on the data and conditional statements.

The data tables must always start with a command row that defines the "Level." The Level associates the table to an element from the XML data file, and establishes the hierarchy. The data fields that are then defined in the table for the Level correspond to the child elements of the XML element.

The graphic below illustrates the relationship between the XML data hierarchy and the template Level. The XML element "RequestHeader" is defined as the Level. The data elements defined in the table ("FileID" and "Encryption") are children of the RequestHeader element.



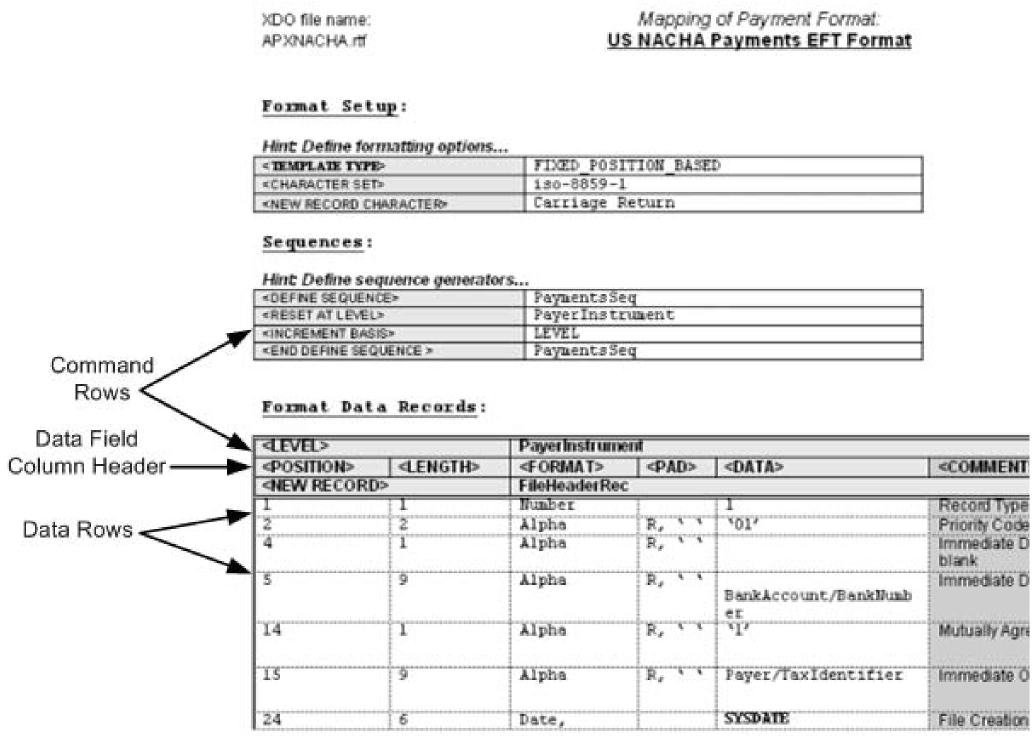
```
?xml version = "1.0" ?>
!DOCTYPE OutBoundPayments:BatchRequest SYSTEM "OutBoundPayments.dtd">
OutBoundPayments:BatchRequest>
  <PaymentsCommon:RequestHeader>
     <PaymentsCommon:Preamble>
        <PaymentsCommon:Version>1.0.11.5.7</PaymentsCommon:Version>
     </PaymentsCommon:Preamble>
     <PaymentsCommon:TrxnParties>
        <PaymentsCommon:RequesterParty>Oracle Germany</PaymentsCommon:Requ
        <PaymentsCommon:FinancialInstitution>Citi Bank</PaymentsCommon:Fir
     </PaymentsCommon:TrxnParties>
     <PaymentsCommon:Encryption>N</PaymentsCommon:Encryption>
     <PaymentsCommon:FileID>10180300001</PaymentsCommon:FileID>
     <PaymentsCommon:BaseRecordCount>94457</PaymentsCommon:BaseRecordCount
  </PaymentsCommon:RequestHeader>

  <OutBoundPayments:B
     <OutBoundPayment
     <OutBoundPayment
     <OutBoundPayment
     <OutBoundPayment
     <OutBoundPayment
     <OutBoundPayment
     <OutBoundPayment
     ...
```

| <LEVEL> | | RequestHeader | | |
|---|---|---|---|---|
| <POSITION> | <LENGTH> | <FORMAT> | <PAD> | <DATA> |
| <NEW RECORD> | | FileHeaderRec | | |
| 1 | 3 | Alpha | | 'F10' |
| 4 | 4 | Number | L, '0' | FileID |
| 8 | 1 | Alpha | R, ' ' | Encryption |
| 9 | 6 | Date, YYMMDD | | **SYSDATE** |
| 15 | 6 | Number | L, '0' | **SEQUENCE_NUMBER(** AllRecordsSeq) |
| 21 | 129 | Alpha | ' ' | |

| <LEVEL> | | Batch | | |
|---|---|---|---|---|
| <POSITION> | <LENGTH> | <FORMAT> | <PAD> | <DATA> |

The order of the tables in the template determines the print order of the records. At runtime the system loops through all the instances of the XML element corresponding to a table (Level) and prints the records belonging to the table. The system then moves on to the next table in the template. If tables are nested, the system will generate the nested records of the child tables before moving on to the next parent instance.

## Command Rows, Data Rows, and Data Column Header Rows



XDO file name:
APXNACHA.rtf

*Mapping of Payment Format:*
**US NACHA Payments EFT Format**

**Format Setup:**

***Hint: Define formatting options...***

| | |
|---|---|
| <TEMPLATE TYPE> | FIXED_POSITION_BASED |
| <CHARACTER SET> | iso-8859-1 |
| <NEW RECORD CHARACTER> | Carriage Return |

**Sequences:**

***Hint: Define sequence generators...***

| | |
|---|---|
| <DEFINE SEQUENCE> | PaymentsSeq |
| <RESET AT LEVEL> | PayerInstrument |
| <INCREMENT BASIS> | LEVEL |
| <END DEFINE SEQUENCE > | PaymentsSeq |

**Format Data Records:**

| <LEVEL> | | PayerInstrument | | | |
|---|---|---|---|---|---|
| <POSITION> | <LENGTH> | <FORMAT> | <PAD> | <DATA> | <COMMENT |
| <NEW RECORD> | | FileHeaderRec | | | |
| 1 | 1 | Number | | 1 | Record Type |
| 2 | 2 | Alpha | R, ' ' | '01' | Priority Code |
| 4 | 1 | Alpha | R, ' ' | | Immediate D blank |
| 5 | 9 | Alpha | R, ' ' | BankAccount/BankNumber | Immediate D |
| 14 | 1 | Alpha | R, ' ' | '1' | Mutually Agre |
| 15 | 9 | Alpha | R, ' ' | Payer/TaxIdentifier | Immediate O |
| 24 | 6 | Date, | | SYSDATE | File Creation |

Command rows are used to specify commands in the template. Command rows always have two columns: command name and command parameter. Command rows do not have column headings. The commands control the overall setup and record structures of the template.

Blank rows can be inserted anywhere in a table to improve readability. Most often they are used in the setup table, between commands. Blank rows are ignored by XML Publisher when the template is parsed.

## Data Column Header Rows

Data column headers specify the column headings for the data fields (such as Position, Length, Format, Padding, and Comments). A column header row usually follows the Level command in a table (or the sorting command, if one is used). The column header row must come before any data rows in the table. Additional empty column header rows can be inserted at any position in a table to improve readability. The empty rows will be ignored at runtime.

The required data column header rows vary depending on the template type. See Structure of the Data Row, page 4-10.

## Data Rows

Data rows contain the data fields to correspond to the column header rows.

The content of the data rows varies depending on the template type. See Structure of the Data Row, page 4-10.

## Constructing the Data Tables

The data tables contain a combination of command rows and data field rows. Each data table must begin with a Level command row that specifies its XML element. Each record must begin with a New Record command that specifies the start of a new record, and the end of a previous record (if any).

The required columns for the data fields vary depending on the Template Type.

## Command Rows

The command rows always have two columns: command name and command parameter. The supported commands are:

- Level
- New record
- Sort ascending
- Sort descending
- Display condition

The usage for each of these commands is described in the following sections.

### Level Command

The level command associates a table with an XML element. The parameter for the level command is an XML element. The level will be printed once for each instance the XML element appears in the data input file.

The level commands define the hierarchy of the template. For example, Payment XML data extracts are hierarchical. A batch can have multiple child payments, and a payment can have multiple child invoices. This hierarchy is represented in XML as nested child elements within a parent element. By associating the tables with XML elements through the level command, the tables will also have the same hierarchical structure.

Similar to the closing tag of an XML element, the level command has a companion end-level command. The child tables must be defined between the level and end-level commands of the table defined for the parent element.

An XML element can be associated with only one level. All the records belonging to a level must reside in the table of that level or within a nested table belonging to that level. The end-level command will be specified at the end of the final table.

Following is a sample structure of an EFT file record layout:

- FileHeaderRecordA
    - BatchHeaderRecordA
    - BatchHeaderRecordB

      PaymentRecordA

      PaymentRecordB
        - InvoiceRecordA
    - Batch FooterRecordC
    - BatchFooterRecordD

- FileFooterRecordB

Following would be its table layout:

| | |
|---|---|
| <LEVEL> | **RequestHeader** |
| <NEW RECORD> | FileHeaderRecordA |
| Data rows for the FileHeaderRecordA | |

| | |
|---|---|
| <LEVEL> | **Batch** |
| <NEW RECORD> | BatchHeaderRecordA |
| Data rows for the BatchHeaderRecordA | |
| <NEW RECORD> | BatchHeaderRecordB |
| Data rows for the BatchHeaderRecordB | |

| | |
|---|---|
| <LEVEL> | **Payment** |
| <NEW RECORD> | PaymentRecordA |
| Data rows for the PaymentRecordA | |
| <NEW RECORD> | PaymentRecordB |
| Data rows for the PaymentRecordB | |

| | |
|---|---|
| <LEVEL> | **Invoice** |
| <NEW RECORD> | InvoiceRecordA |
| Data rows for the InvoiceRecordA | |
| <END LEVEL> | **Invoice** |

| | |
|---|---|
| <END LEVEL> | **Payment** |

| | |
|---|---|
| <LEVEL> | **Batch** |
| <NEW RECORD> | BatchFooterRecordC |
| Data rows for the BatchFooterRecordC | |
| <NEW RECORD> | BatchFooterRecordD |
| Data rows for the BatchFooterRecordD | |
| <END LEVEL> | **Batch** |

| | |
|---|---|
| <LEVEL> | **RequestHeader** |
| <NEW RECORD> | FileFooterRecordB |
| Data rows for the FileFooterRecordB | |
| <END LEVEL> | **RequestHeader** |

Multiple records for the same level can exist in the same table. However, each table can only have one level defined. In the example above, the BatchHeaderRecordA and BatchHeaderRecordB are both defined in the same table. However, note that the END LEVEL for the Payment must be defined in its own separate table after the child element Invoice. The Payment END LEVEL cannot reside in the same table as the Invoice Level.

Note that you do not have to use all the levels from the data extract in your template. For example, if an extract contains the levels: RequestHeader > Batch > Payment > Invoice, you can use just the batch and invoice levels. However, the hierarchy of the levels must be maintained.

The table hierarchy determines the order that the records are printed. For each parent XML element, the records of the corresponding parent table are printed in the order they appear in the table. The system loops through the instances of the child XML elements corresponding to the child tables and prints the child records according to their specified order. The system then prints the records of the enclosing (end-level) parent table, if any.

For example, given the EFT template structure above, assume the input data file contains the following:

- Batch1
  - Payment1
    - Invoice1
    - Invoice2
  - Payment2
    - Invoice1
- Batch2
  - Payment1
    - Invoice1

- Invoice2
- Invoice3

This will generate the following printed records:

| Record Order | Record Type | Description |
|---|---|---|
| 1 | FileHeaderRecordA | One header record for the EFT file |
| 2 | BatchHeaderRecordA | For Batch1 |
| 3 | BatchHeaderRecordB | For Batch1 |
| 4 | PaymentRecordA | For Batch1, Payment1 |
| 5 | PaymentRecordB | For Batch1, Payment1 |
| 6 | InvoiceRecordA | For Batch1, Payment1, Invoice1 |
| 7 | InvoiceRecordA | For Batch1, Payment1, Invoice2 |
| 8 | PaymentRecordA | For Batch1, Payment2 |
| 9 | PaymentrecordB | For Batch1, Payment2 |
| 10 | InvoiceRecordA | For Batch1, Payment2, Invoice1 |
| 11 | BatchFooterRecordC | For Batch1 |
| 12 | BatchFooterRecordD | For Batch1 |
| 13 | BatchHeaderRecordA | For Batch2 |
| 14 | BatchHeaderRecordB | For Batch2 |
| 15 | PaymentRecordA | For Batch2, Payment1 |
| 16 | PaymentRecordB | For Batch2, Payment1 |
| 17 | InvoiceRecordA | For Batch2, Payment1, Invoice1 |
| 18 | InvoiceRecordA | For Batch2, Payment1, Invoice2 |
| 19 | InvoiceRecordA | For Batch2, Payment1, Invoice3 |
| 20 | BatchFooterRecordC | For Batch2 |
| 21 | BatchFooterRecordD | For Batch2 |
| 22 | FileFooterRecordB | One footer record for the EFT file |

### New Record Command

The new record command signifies the start of a record and the end of the previous one, if any. Every record in a template must start with the new record command. The record continues until the next new record command, or until the end of the table or the end of the level command.

A record is a construct for the organization of the elements belonging to a level. The record name is not associated with the XML input file.

A table can contain multiple records, and therefore multiple new record commands. All the records in a table are at the same hierarchy level. They will be printed in the order in which they are specified in the table.

The new record command can have a name as its parameter. This name becomes the name for the record. The record name is also referred to as the record type. The name can be used in the COUNT function for counting the generated instances of the record. See COUNT, page 4-24 function, for more information.

Consecutive new record commands (or empty records) are not allowed.

### Sort Ascending and Sort Descending Commands

Use the sort ascending and sort descending commands to sort the instances of a level. Enter the elements you wish to sort by in a comma-separated list. This is an optional command. When used, it must come right after the (first) level command and it applies to all records of the level, even if the records are specified in multiple tables.

### Display Condition Command

The display condition command specifies when the enclosed record or data field group should be displayed. The command parameter is a boolean expression. When it evaluates to true, the record or data field group is displayed. Otherwise the record or data field group is skipped.

The display condition command can be used with either a record or a group of data fields. When used with a record, the display condition command must follow the new record command. When used with a group of data fields, the display condition command must follow a data field row. In this case, the display condition will apply to the rest of the fields through the end of the record.

Consecutive display condition commands are merged as AND conditions. The merged display conditions apply to the same enclosed record or data field group.

## Structure of the Data Rows

The output record data fields are represented in the template by table rows. In FIXED_POSITION_BASED templates, each row has the following attributes (or columns):

- Position
- Length
- Format
- Pad
- Data
- Comments

The first five columns are required and must appear in the order listed.

For DELIMITER_BASED templates, each data row has the following attributes (columns):

- Maximum Length
- Format
- Data
- Tag
- Comments

The first three columns are required and must be declared in the order stated.

In both template types, the Comments column is optional and ignored by the system. You can insert additional information columns if you wish, as all columns after the required ones are ignored.

The usage rules for these columns are as follows:

### Position

Specifies the starting position of the field in the record. The unit is in number of characters. This column is only used with FIXED_POSITION_BASED templates.

### Length/Maximum Length

Specifies the length of the field. The unit is in number of characters. For FIXED_POSITION_BASED templates, all the fields are fixed length. If the data is less than the specified length, it is padded. If the data is longer, it is truncated. The truncation always occurs on the right.

For DELIMITER_BASED templates, the maximum length of the field is specified. If the data exceeds the maximum length, it will be truncated. Data is not padded if it is less than the maximum length.

### Format

Specifies the data type and format setting. There are three accepted data types: Alpha, Number, and Date. Refer to Field Level Key Words, page 4-26 for their usage.

Numeric data has two optional format settings: Integer and Decimal. Specify the optional settings with the Number data type as follows:

- Number, Integer
- Number, Decimal

The Integer format uses only the whole number portion of a numeric value and discards the decimal. The Decimal format uses only the decimal portion of the numeric value and discards the integer portion.

The Date data type format setting must always be explicitly stated. The format setting follows the SQL date styles, such as MMDDYY.

Some EDI (DELIMITER_BASED) formats use more descriptive data types. These are mapped to the three template data types in the following table:

| ASC X12 Data Type | Format Template Data Type |
|---|---|
| A - Alphabetic | Alpha |
| AN -Alphanumeric | Alpha |
| B - Binary | Number |
| CD - Composite data element | N/A |
| CH - Character | Alpha |
| DT - Date | Date |
| FS - Fixed-length string | Alpha |
| ID - Identifier | Alpha |
| IV - Incrementing Value | Number |
| Nn - Numeric | Number |
| PW - Password | Alpha |
| R - Decimal number | Numer |
| TM - Time | Date |

### Pad

This applies to FIXED_POSITION_BASED templates only. Specify the padding side (L = left or R = right) and the character. Both numeric and alphanumeric fields can be padded. If this field is not specified, Numeric fields are left-padded with "0"; Alpha fields are right-padded with spaces.

Example usage:

- To pad a field on the left with a "0", enter the following in the Pad column field:

  L, ’0’

- To pad a field on the right with a space, enter the following the Pad column field:

  R, ’ ’

### Data

Specifies the XML element from the data extract that is to populate the field. The data column can simply contain the XML tag name, or it can contain expressions and functions. For more information, see Expressions, Control Structure, and Functions, page 4-23.

### Tag

Acts as a comment column for DELIMITER_BASED templates. It specifies the reference tag in EDIFACT formats, and the reference IDs in ASC X12.

#### Comments

Use this column to note any free form comments to the template. Usually this column is used to note the business requirement and usage of the data field.

## Setup Command Tables

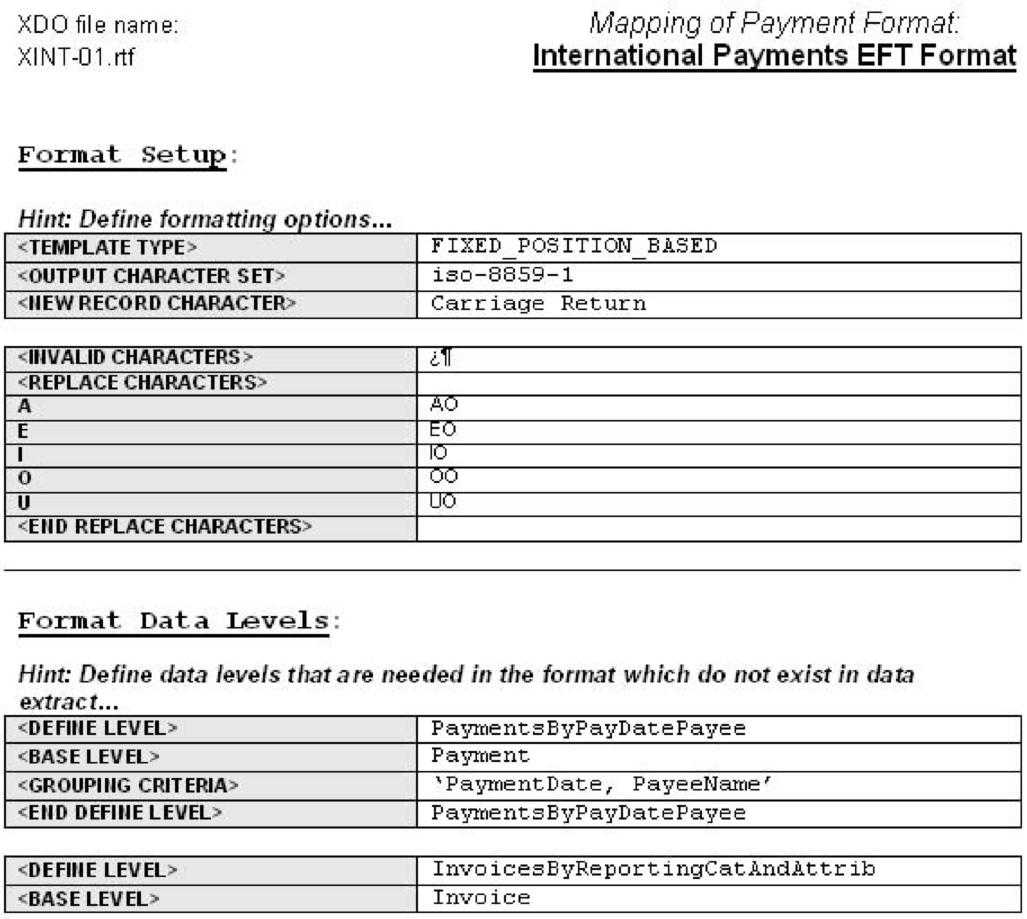
### Setup Command Table

A template always begins with a table that specifies the setup commands. The setup commands define global attributes, such as template type and output character set and program elements, such as sequencing and concatenation.

The setup commands are:

- Template Type
- Output Character Set
- New Record Character
- Invalid Characters
- Replace Characters
- Define Level
- Define Sequence
- Define Concatenation

Some example setup tables are shown in the following figures:



XDO file name:
XINT-01.rtf

*Mapping of Payment Format:*
**International Payments EFT Format**

**Format Setup:**

*Hint: Define formatting options...*

| | |
|---|---|
| **<TEMPLATE TYPE>** | `FIXED_POSITION_BASED` |
| **<OUTPUT CHARACTER SET>** | `iso-8859-1` |
| **<NEW RECORD CHARACTER>** | `Carriage Return` |

| | |
|---|---|
| **<INVALID CHARACTERS>** | ¿¶ |
| **<REPLACE CHARACTERS>** | |
| **A** | AO |
| **E** | EO |
| **I** | IO |
| **O** | OO |
| **U** | UO |
| **<END REPLACE CHARACTERS>** | |

**Format Data Levels:**

*Hint: Define data levels that are needed in the format which do not exist in data extract...*

| | |
|---|---|
| **<DEFINE LEVEL>** | `PaymentsByPayDatePayee` |
| **<BASE LEVEL>** | `Payment` |
| **<GROUPING CRITERIA>** | `‘PaymentDate, PayeeName’` |
| **<END DEFINE LEVEL>** | `PaymentsByPayDatePayee` |

| | |
|---|---|
| **<DEFINE LEVEL>** | `InvoicesByReportingCatAndAttrib` |
| **<BASE LEVEL>** | `Invoice` |

| | |
|---|---|
| <GROUPING CRITERIA> | 'InvoiceTrxnReportingCat', (IF InvoiceTrxnReportingCat = 'V' THEN 'InvoiceDEVTransitGoods' END IF), (IF InvoiceTrxnReportingCat = 'V' THEN 'InvoiceDEVGoodsIndexNum' END IF), (IF InvoiceTrxnReportingCat = 'V' THEN 'InvoiceDEVPassingTrade' END IF) |
| <END DEFINE LEVEL> | InvoicesByReportingCatAndAttrib |

Sequences:

*Hint: Define sequence generators...*

| | |
|---|---|
| <DEFINE SEQUENCE> | AllRecordsSeq |
| <RESET AT LEVEL> | PERIODIC_SEQUENCE |
| <INCREMENT BASIS> | RECORD |
| <START AT> | BaseRecordCount + 1 |
| <MAXIMUM> | 999999 |
| <END DEFINE SEQUENCE > | AllRecordsSeq |

| | |
|---|---|
| <DEFINE SEQUENCE> | PaymentsSeq |
| <RESET AT LEVEL> | Batch |
| <INCREMENT BASIS> | LEVEL |
| <END DEFINE SEQUENCE > | PaymentsSeq |

Concatenated Records:

*Hint: Define fields that are composed of concatenated records...*

| | |
|---|---|
| <DEFINE CONCATENATION> | ConcatenatedInvoiceInfo |
| <BASE LEVEL> | Invoice |
| <ELEMENT> | InvoiceNum |
| <DELIMITER> | ',' |
| <END DEFINE CONCATENATION> | ConcatenatedInvoiceInfo |

## Template Type Command

This command specifies the type of template. There are two types: FIXED_POSIT ION_BASED and DELIMITER_BASED.

Use the FIXED_POSITION_BASED templates for fixed-length record formats, such as EFTs. In these formats, all fields in a record are a fixed length. If data is shorter than the specified length, it will be padded. If longer, it will be truncated. The system specifies the default behavior for data padding and truncation. Examples of fixed position based formats are EFTs in Europe, and NACHA ACH file in the U.S.

In a DELIMITER_BASED template, data is never padded and only truncated when it has reached a maximum field length. EMpty fields are allowed (when the data is null). Designated delimiters are used to separate the data fields. If a field is empty, two delimiters will appear next to each other. Examples of delimited-based templates are EDI formats such as ASC X12 820 and UN EDIFACT formats - PAYMUL, DIRDEB, and CREMUL.

In EDI formats, a record is sometimes referred to as a segment. An EDI segment is treated the same as a record. Start each segment with a new record command and give it a record name. You should have a data field specifying the segment name as part of the output data immediately following the new record command.

For DELIMITER_BASED templates, you insert the appropriate data field delimiters in separate rows between the data fields. After every data field row, you insert a delimiter row. You can insert a placeholder for an empty field by defining two consecutive delimiter rows.

Empty fields are often used for syntax reasons: you must insert placeholders for empty fields so that the fields that follow can be properly identified.

There are different delimiters to signify data fields, composite data fields, and end of record. Some formats allow you to choose the delimiter characters. In all cases you should use the same delimiter consistently for the same purpose to avoid syntax errors.

In DELIMITER_BASED templates, the <POSITION> and <PAD> columns do not apply. They are omitted from the data tables.

Some DELIMITER_BASED templates have minimum and maximum length specifications. In those cases Oracle Payments validates the length.

### Define Level Command

Some formats require specific additional data levels that are not in the data extract. For example, some formats require that payments be grouped by payment date. Using the Define Level command, a payment date group can be defined and referenced as a level in the template, even though it is not in the input extract file.

When you use the Define Level command you declare a base level that exists in the extract. The Define Level command inserts a new level one level higher than the base level of the extract. The new level functions as a grouping of the instances of the base level.

The Define Level command is a setup command, therefore it must be defined in the setup table. It has two subcommands:

- Base Level Command - defines the level (XML element) from the extract that the new level is based on. The Define Level command must always have one and only one base level subcommand.

  Grouping Criteria - defines the XML extract elements that are used to group the instances of the base level to form the instances of the new level. The parameter of the grouping criteria command is a comma-separated list of elements that specify the grouping conditions.

  The order of the elements determines the hierarchy of the grouping. The instances of the base level are first divided into groups according to the values of the first criterion, then each of these groups is subdivided into groups according to the second criterion, and so on. Each of the final subgroups will be considered as an instance of the new level.

For example, the following table shows five payments under a batch:

| Payment Instance | PaymentDate (grouping criterion 1) | PayeeName (grouping criterion 2) |
|---|---|---|
| Payment1 | PaymentDate1 | PayeeName1 |
| Payment2 | PaymentDate2 | PayeeName1 |
| Payment3 | PaymentDate1 | PayeeName2 |
| Payment4 | PaymentDate1 | PayeeName1 |
| Payment5 | PaymentDate1 | PayeeName3 |

In the template, construct the setup table as follows to create a level called "PaymentsByPayDatePayee" from the base level "Payment" grouped according to PaymentDate and Payee Name:

| | |
|---|---|
| <DEFINE LEVEL> | PaymentsByPayDatePayee |
| <BASE LEVEL> | Payment |
| <GROUPING CRITERIA> | PaymentDate, PayeeName |
| <END DEFINE LEVEL> | PaymentsByPayDatePayee |

The five payments will generate the following four groups (instances) for the new level:

| Payment Group Instance | Group Criteria | Payments in Group |
|---|---|---|
| Group1 | PaymentDate1, PayeeName1 | Payment1, Payment4 |
| Group2 | PaymentDate1, PayeeName2 | Payment3 |
| Group3 | PaymentDate1, PayeeName3 | Payment5 |
| Group4 | PaymentDate2, PayeeName1 | Payment2 |

The order of the new instances is the order that the records will print. When evaluating the multiple grouping criteria to form the instances of the new level, the criteria can be thought of as forming a hierarchy. The first criterion is at the top of the hierarchy, the last criterion is at the bottom of the hierarchy.

Generally there are two kinds of format-specific data grouping scenarios in EFT formats. Some formats print the group records only; others print the groups with the individual element records nested inside groups. Following are two examples for these scenarios based on the five payments and grouping conditions previously illustrated.

**Example**

First Scenario: Group Records Only

EFT File Structure:

- BatchRec

    - PaymentGroupHeaderRec
    - PaymentGroupFooterRec

| Record Sequence | Record Type | Description |
|---|---|---|
| 1 | BatchRec | |
| 2 | PaymentGroupHeaderRec | For group 1 (PaymentDate1, PayeeName1) |
| 3 | PaymentGroupFooterRec | For group 1 (PaymentDate1, PayeeName1) |
| 4 | PaymentGroupHeaderRec | For group 2 (PaymentDate1, PayeeName2) |
| 5 | PaymentGroupFooterRec | For group 2 (PaymentDate1, PayeeName2) |
| 6 | PaymentGroupHeaderRec | For group 3 (PaymentDate1, PayeeName3) |
| 7 | PaymentGroupFooterRec | For group 3 (PaymentDate1, PayeeName3) |
| 8 | PaymentGroupHeaderRec | For group 4 (PaymentDate2, PayeeName1) |
| 9 | PaymentGroupFooterRec | For group 4 (PaymentDate2, PayeeName1) |

### Example

Scenario 2: Group Records and Individual Records

EFT File Structure:

BatchRec

- PaymentGroupHeaderRec
    - PaymentRec
- PaymentGroupFooterRec

Generated output:

| Record Sequence | Record Type | Description |
|---|---|---|
| 1 | BatchRec | |
| 2 | PaymentGroupHeaderRec | For group 1 (PaymentDate1, PayeeName1) |
| 3 | PaymentRec | For Payment1 |
| 4 | PaymentRec | For Payment4 |
| 5 | PaymentGroupFooterRec | For group 1 (PaymentDate1, PayeeName1) |
| 6 | PaymentGroupHeaderRec | For group 2 (PaymentDate1, PayeeName2) |
| 7 | PaymentRec | For Payment3 |
| 8 | PaymentGroupFooterRec | For group 2 (PaymentDate1, PayeeName2) |
| 9 | PaymentGroupHeaderRec | For group 3 (PaymentDate1, PayeeName3) |
| 10 | PaymentRec | For Payment5 |
| 11 | PaymentGroupFooterRec | For group 3 (PaymentDate1, PayeeName3) |
| 12 | PaymentGroupHeaderRec | For group 4 (PaymentDate2, PayeeName1) |
| 13 | PaymentRec | For Payment2 |
| 14 | PaymentGroupFooterRec | For group 4 (PaymentDate2, PayeeName1) |

Once defined with the Define Level command, the new level can be used in the template in the same manner as a level occurring in the extract. However, the records of the new level can only reference the base level fields that are defined in its grouping criteria. They cannot reference other base level fields other than in summary functions.

For example, the PaymentGroupHeaderRec can reference the PaymentDate and PayeeName in its fields. It can also reference thePaymentAmount (a payment level field) in a SUM function. However, it cannot reference other payment level fields, such as PaymentDocName or PaymentDocNum.

The Define Level command must always have one and only one grouping criteria subcommand. The Define Level command has a companion end-define level command. The subcommands must be specified between the define level and end-define level commands. They can be declared in any order.

## Define Sequence Command

The define sequence command define a sequence that can be used in conjunction with the SEQUENCE_NUMBER function to index either the generated EFT records or the extract instances (the database records). The EFT records are the physical records defined in the template. The database records are the records from the extract. To avoid confusion, the term "record" will always refer to the EFT record. The database record will be referred to as an extract element instance or level.

The define sequence command has four subcommands: reset at level, increment basis, start at, and maximum:

### Reset at Level

The reset at level subcommand defines where the sequence resets its starting number. It is a mandatory subcommand. For example, to number the payments in a batch, define the reset at level as Batch. To continue numbering across batches, define the reset level as RequestHeader.

In some cases the sequence is reset outside the template. For example, a periodic sequence may be defined to reset by date. In these cases, the PERIODIC_SEQUENCE keyword is used for the reset at level. The system saves the last sequence number used for a payment file to the database. Outside events control resetting the sequence in the database. For the next payment file run, the sequence number is extracted from the database for the start at number (see start at subcommand).

### Increment Basis

The increment basis subcommand specifies if the sequence should be incremented based on record or extract instances. The allowed parameters for this subcommand are RECORD and LEVEL.

Enter RECORD to increment the sequence for every record.

Enter LEVEL to increment the sequence for every new instance of a level.

Note that for levels with multiple records, if you use the level-based increment all the records in the level will have the same sequence number. The record-based increment will assign each record in the level a new sequence number.

For level-based increments, the sequence number can be used in the fields of one level only. For example, suppose an extract has a hierarchy of batch > payment > invoice and you define the increment basis by level sequence, with reset at the batch level. You can use the sequence in either the payment or invoice level fields, but not both. You cannot have sequential numbering across hierarchical levels.

However, this rule does not apply to increment basis by record sequences. Records can be sequenced across levels.

For both increment basis by level and by record sequences, the level of the sequence is implicit based on where the sequence is defined.

## Define Concatenation Command

Use the define concatenation command to concatenate child-level extract elements for use in parent-level fields. For example, use this command to concatenate invoice number and due date for all the invoices belonging to a payment for use in a payment-level field.

The define concatenation command has three subcommands: base level, element, and delimiter.

#### Base Level Subcommand

The base level subcommand specifies the child level for the operation. For each parent-level instance, the concatenation operation loops through the child-level instances to generate the concatenated string.

#### Item Subcommand

The item subcommand specifies the operation used to generate each item. An item is a child-level expression that will be concatenated together to generate the concatenation string.

#### Delimiter Subcommand

The delimiter subcommand specifies the delimiter to separate the concatenated items in the string.

#### Using the SUBSTR Function

Use the SUBSTR function to break down concatenated strings into smaller strings that can be placed into different fields. For example, the following table shows five invoices in a payment:

| Invoice | InvoiceNum |
|---|---|
| 1 | car_parts_inv0001 |
| 2 | car_parts_inv0002 |
| 3 | car_parts_inv0003 |
| 4 | car_parts_inv0004 |
| 5 | car_parts_inv0005 |

Using the following concatenation definition:

| | |
|---|---|
| <DEFINE CONCATENATION> | ConcatenatedInvoiceInfo |
| <BASE LEVEL> | Invoice |
| <ELEMENT> | InvoiceNum |
| <DELIMITER> | ',' |
| <END DEFINE CONCATENATION> | ConcatenatedInvoiceInfo |

You can reference ConcatenatedInvoiceInfo in a payment level field. The string will be:

```
car_parts_inv0001,car_parts_inv0002,car_parts_inv0003,car_
parts_inv0004,car_parts_inv0005
```

If you want to use only the first forty characters of the concatenated invoice info, use eith ther TRUNCATE function or the SUBSTR function as follows:

```
TRUNCATE(ConcatenatedInvoiceInfo, 40)
```

```
SUBSTR(ConctenatedInvoiceInfo, 1, 40)
```

Either of these statements will result in:

**`car_parts_inv0001,car_parts_inv0002,car_`**

To isolate the next forty characters, use the SUBSTR function:

**`SUBSTR(ConcatenatedInvoiceInfo, 41, 40)`**

to get the following string:

**`parts_inv0003,car_parts_inv0004,car_par`**

## Invalid Characters and Replacement Characters Commands

Some formats require a different character set than the one that was used to enter the data in Oracle Applications. For example, some German formats require the output file in ASCII, but the data was entered in German. If there is a mismatch between the original and target character sets you can define an ASCII equivalent to replace the original. For example, you would replace the German umlauted "a" with "ao".

Some formats will not allow certain characters. To ensure that known invalid characters will not be transmitted in your output file, use the invalid characters command to flag occurrences of specific characters.

To use the replacement characters command, specify the source characters in the left column and the replacement characters in the right column. You must enter the source characters in the original character set. This is the only case in a format template in which you use a character set not intended for output. Enter the replacement characters in the required output character set.

For DELIMITER_BASED formats, if there are delimiters in the data, you can use the escape character "?" to retain their meaning. For example,

First name?+Last name equates to Fist name+Last name

Which source?? equates to Which source?

Note that the escape character itself must be escaped if it is used in data.

The replacement characters command can be used to support the escape character requirement. Specify the delimiter as the source and the escape character plus the delimiter as the target. For example, the command entry for the preceding examples would be:

| | |
|---|---|
| <REPLACEMENT CHARACTERS> | |
| + | ?+ |
| ? | ?? |
| <END REPLACEMENT CHARACTERS> | |

The invalid character command has a single parameter that is a string of invalid characters that will cause the system to error out.

The replacement character process is performed before or during the character set conversion. The character set conversion is performed on the XML extract directly, before the formatting. After the character set conversion, the invalid characters will be checked in terms of the output character set. If no invalid characters are found, the system will proceed to formatting.

#### Output Character Set and New Record Character Commands

Use the new record character command to specify the character(s) to delimit the explicit and implicit record breaks at runtime. Each new record command represents an explicit record break. Each end of table represents an implicit record break. The parameter is a list of constant character names separated by commas.

Some formats contain no record breaks. The generated output is a single line of data. In this case, leave the new record character command parameter field empty.

## Expressions, Control Structure, and Functions

This section describes the rules and usage for expressions in the template. It also describes supported control structures and functions.

### Expressions

Expressions can be used in the data column for data fields and some command parameters. An expression is a group of XML extract fields, literals, functions, and operators. Expressions can be nested. An expression can also include the "IF" control structure. When an expression is evaluated it will always generate a result. Side effects are not allowed for the evaluation. Based on the evaluation result, expressions are classified into the following three categories:

- Boolean Expression - an expression that returns a boolean value, either true or false. This kind expression can be used only in the "IF-THEN-ELSE" control structure and the parameter of the display condition command.
- Numeric Expression - an expression that returns a number. This kind of expression can be used in numeric data fields. It can also be used in functions and commands that require numeric parameters.
- Character Expression - an expression that returns an alphanumeric string. This kind of expression can be used in string data fields (format type Alpha). They can also be used in functions and command that require string parameters.

### Control Structures

The only supported control structure is "IF-THEN-ELSE". It can be used in an expression. The syntax is:

```
IF <boolean_expressionA> THEN
   <numeric or character expression1>
[ELSIF <boolean_expressionB THEN
   <numeric or character expression2>]
...
[ELSE
   <numeric or character expression2]
END IF
```

Generally the control structure must evaluate to a number or an alphanumeric string. The control structure is considered to a numeric or character expression. The ELSIF and ELSE clauses are optional, and there can be as many ELSIF clauses as necessary. The control structure can be nested.

### Functions

Following is the list of supported functions:

- SEQUENCE_NUMBER - is a record element index. It is used in conjunction with the Define Sequence command. It has one parameter, which is the sequence defined by the Define Sequence command. At runtime it will increase its sequence value by one each time it is referenced in a record.
- COUNT - counts the child level extract instances or child level records of a specific type. Declare the COUNT function on a level above the entity to be counted. The function has one argument. If the argument is a level, the function will count all the instances of the (child) level belonging to the current (parent) level instance.

  For example, if the level to be counted is Payment and the current level is Batch, then the COUNT will return the total number of payments in the batch. However, if the current level is RequestHeader, the COUNT will return the total number of payments in the file across all batches. If the argument is a record type, the count function will count all the generated records of the (child level) record type belonging to the current level instance.
- INTEGER_PART, DECIMAL_PART - returns the integer or decimal portion of a numeric value. This is used in nested expressions and in commands (display condition and group by). For the final formatting of a numeric field in the data column, use the Integer/Decimal format.
- IS_NUMERIC - boolean test whether the argument is numeric. Used only with the "IF" control structure.
- TRUNCATE - truncate the first argument - a string to the length of the second argument. If the first argument is shorter than the length specified by the second argument, the first argument is returned unchanged. This is a user-friendly version for a subset of the SQL substr() functionality.
- SUM - sums all the child instance of the XML extract field argument. The field must be a numeric value. The field to be summed must always be at a lower level than the level on which the SUM function was declared.
- MIN, MAX - find the minimum or maximum of all the child instances of the XML extract field argument. The field must be a numeric value. The field to be operated on must always be at a lower level than the level on which the function was declared.
- Other SQL functions: TO_DATE, LOWER, UPPER, LENGTH, GREATEST, LEAST - use the syntax corresponding to the SQL function.

## Identifiers, Operators, and Literals

This section lists the reserved key word and phrases and their usage. The supported operators are defined and the rules for referencing XML extract fields and using literals.

### Key Words

There are four categories of key words and key word phrases:

- Command and column header key words
- Command parameter and function parameter key words
- Field-level key words
- Expression key words

## Command and Column Header Key Words

The following key words must be used as shown: enclosed in <>s and in all capital letters with a bold font.

- **<LEVEL>** - the first entry of a data table. Associates the table with an XML element and specifies the hierarchy of the table.
- **<END LEVEL>** - declares the end of the current level. Can be used at the end of a table or in a standalone table.
- **<POSITION>** - column header for the first column of data field rows, which specifies the starting position of the data field in a record.
- **<LENGTH>** - column header for the second column of data field rows, which specifies the length of the data field.
- **<FORMAT>** - column header for the third column of data field rows, which specifies the data type and format setting.
- **<PAD>** - column header for the fourth column of data field rows, which specifies the padding style and padding character.
- **<DATA>** - column header for the fifth column of data field rows, which specifies the data source.
- **<COMMENT>** - column header for the sixth column of data field rows, which allows for free form comments.
- **<NEW RECORD>** - specifies a new record.
- **<DISPLAY CONDITION>** - specifies the condition when a record should be printed.
- **<TEMPLATE TYPE>** - specifies the type of the template, either FIXED_POSITION_BASED or DELIMITER_BASED.
- **<OUTPUT CHARACTER SET>** - specifies the character set to be used when generating the output.
- **<NEW RECORD CHARACTER>** - specifies the character(s) to use to signify the explicit and implicit new records at runtime.
- **<DEFINE LEVEL>** - defines a format-specific level in the template.
- **<BASE LEVEL>** - subcommand for the define level and define concatenation commands.
- **<GROUPING CRITERIA>** - subcommand for the define level command.
- **<END DEFINE LEVEL>** - signifies the end of a level.
- **<DEFINE SEQUENCE>** - defines a record or extract element based sequence for use in the template fields.
- **<RESET AT LEVEL>** - subcommand for the define sequence command.
- **<INCREMENT BASIS>** - subcommand for the define sequence command.
- **<START AT>** - subcommand for the define sequence command.
- **<MAXIMUM>** - subcommand for the define sequence command.
- **<MAXIMUM LENGTH>** - column header for the first column of data field rows, which specifies the maximum length of the data field. For DELIMITER_BASED templates only.

- **<END DEFINE SEQUENCE>** - signifies the end of the sequence command.
- **<DEFINE CONCATENATION>** - defines a concatenation of child level item that can be referenced as a string the parent level fields.
- **<ELEMENT>** - subcommand for the define concatenation command.
- **<DELIMITER>** - subcommand for the define concatenation command.
- **<END DEFINE CONCATENATION>** - signifies the end of the define concatenation command.
- **<SORT ASCENDING>** - format-specific sorting for the instances of a level.
- **<SORT DESCENDING>** - format-specific sorting for the instances of a level.

### Command Parameter and Function Parameter Key Words

These key words must be entered in all capital letters, nonbold fonts.

- PERIODIC_SEQUENCE - used in the reset at level subcommand of the define sequence command. It denotes that the sequence number is to be reset outside the template.
- FIXED_POSITION_BASED, DELIMITER_BASED - used in the template type command, specifies the type of template.
- RECORD, LEVEL - used in the increment basis subcommand of the define sequence command. RECORD increments the sequence each time it is used in a new record. LEVEL increments the sequence only for a new instance of the level.

### Field-Level Key Words

- Alpha - in the <FORMAT> column, specifies the data type is alphanumeric.
- Number - in the <FORMAT> column, specifies the data type is numeric.
- Integer - in the <FORMAT> column, used with the Number key word. Takes the integer part of the number. This has the same functionality as the INTEGER function, except the INTEGER function is used in expressions, while the Integer key word is used in the <FORMAT> column only.
- Decimal - in the <FORMAT> column, used with the Number key word. Takes the decimal part of the number. This has the same functionality as the DECIMAL function, except the DECIMAL function is used in expressions, while the Decimal key word is used in the <FORMAT> column only.
- Date - in the <FORMAT> column, specifies the data type is date.
- L, R- in the <PAD> column, specifies the side of the padding (Left or Right).

### Expression Key Words

Key words and phrases used in expressions must be in capital letters and bold fonts.

- IF THEN ELSE IF THEN ELSE END IF - these key words are always used as a group. They specify the "IF" control structure expressions.
- IS NULL, IS NOT NULL - these phrases are used in the IF control structure. They form part of boolean predicates to test if an expression is NULL or not NULL.

### Operators

There are two groups of operators: the boolean test operators and the expression operators. The boolean test operators include: "=", "<>", "<", ">", ">=", and "<=". They can be used only with the IF control structure. The expression operators include: "()", "| |", "+", "-", and "*". They can be used in any expression.

| Symbol | Usage |
|---|---|
| = | Equal to test. Used in the IF control structure only. |
| <> | Not equal to test. Used in the IF control structure only. |
| > | Greater than test. Used in the IF control structure only. |
| < | Less than test. Used in the IF control structure only. |
| >= | Greater than or equal to test. Used in the IF control structure only. |
| <= | Less than or equal to test. Used in the IF control structure only. |
| () | Function argument and expression group delimiter. The expression group inside "()" will always be evaluated first. "()" can be nested. |
| \| \| | String concatenation operator. |
| + | Addition operator. Implicit type conversion may be performed if any of the operands are not numbers. |
| - | Subtraction operator. Implicit type conversion may be performed if any of the operands are not numbers. |
| * | Multiplication operator. Implicit type conversion may be performed if any of the operands are not numbers. |
| DIV | Division operand. Implicit type conversion may be performed if any of the operands are not numbers. Note that "/" is not used because it is part of the XPATH syntax. |

### Reference to XML Extract Fields and XPATH Syntax

XML elements can be used in any expression. At runtime they will be replaced with the corresponding field values. The field names are case-sensitive.

When the XML extract fields are used in the template, they must follow the XPATH syntax. This is required so that the XML Publisher engine can correctly interpret the XML elements.

There is always an extract element considered as the context element during the XML Publisher formatting process. When XML Publisher processes the data rows in a table, the level element of the table is the context element. For example, when XML Publisher processes the data rows in the Payment table, Payment is the context element. The relative XPATH you use to reference the extract elements are specified in terms of the context element.

For example if you need to refer to the PayeeName element in a Payment data table, you will specify the following relative path:

Payee/PayeeInfo/PayeeName

Each layer of the XML element hierarchy is separated by a backslash "/". You use this notation for any nested elements. The relative path for the immediate child element of the level is just the element name itself. For example, you can use TransactionID element name as is in the Payment table.

To reference a parent level element in a child level table, you can use the "../" notation. For example, in the Payment table if you need to reference the BatchName element, you can specify ../BatchName. The "../" will give you Batch as the context; in that context you can use the BatchName element name directly as BatchName is an immediate child of Batch. This notation goes up to any level for the parent elements. For example if you need to reference the RequesterParty element (in the RequestHeader) in a Payment data table, you can specify the following:

../../TrxnParties/RequesterParty

You can always use the absolute path to reference any extract element anywhere in the template. The absolute path starts with a backslash "/". For the PayeeName in the Payment table example above, you will have the following absolute path: /BatchRequest/Batch/Payment/Payee/PayeeInfo/PayeeName

The absolute path syntax provides better performance.

The identifiers defined by the setup commands such as define level, define sequence and define concatenation are considered to be global. They can be used anywhere in the template. No absolute or relative path is required. The base level and reset at level for the setup commands can also be specified. XML Publisher will be able to find the correct context for them.

If you use relative path syntax, you should specify it relative to the base levels in the following commands:

- The element subcommand of the define concatenation command
- The grouping criteria subcommand of the define level command

The extract field reference in the start at subcommand of the define sequence command should be specified with an absolute path.

The rule to reference an extract element for the level command is the same as the rule for data fields. For example, if you have a Batch level table and a nested Payment level table, you can specify the Payment element name as-is for the Payment table. Because the context for evaluating the Level command of the Payment table is the Batch.

However, if you skip the Payment level and you have an Invoice level table directly under the Batch table, you will need to specify Payment/Invoice as the level element for the Invoice table.

The XPATH syntax required by the template is very similar to UNIX/LINUX directory syntax. The context element is equivalent to the current directory. You can specify a file relative to the current directory or you can use the absolute path which starts with a "/".

Finally, the extract field reference as the result of the grouping criteria sub-command of the define level command must be specified in single quotes. This tells the XML Publisher engine to use the extract fields as the grouping criteria, not their values.

# 5 Using the Template Manager

This chapter covers the following topics:

- Introduction
- Creating the Data Definition
- Creating the Template
- Viewing and Updating a Template

## 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.
- Preview your template with sample data.

To create a template in the Template Manager:

1. Create the data definition for your template, page 5- 1 .
2. Register the layout template file, page 5- 3 .

### Accessing the Template Manager

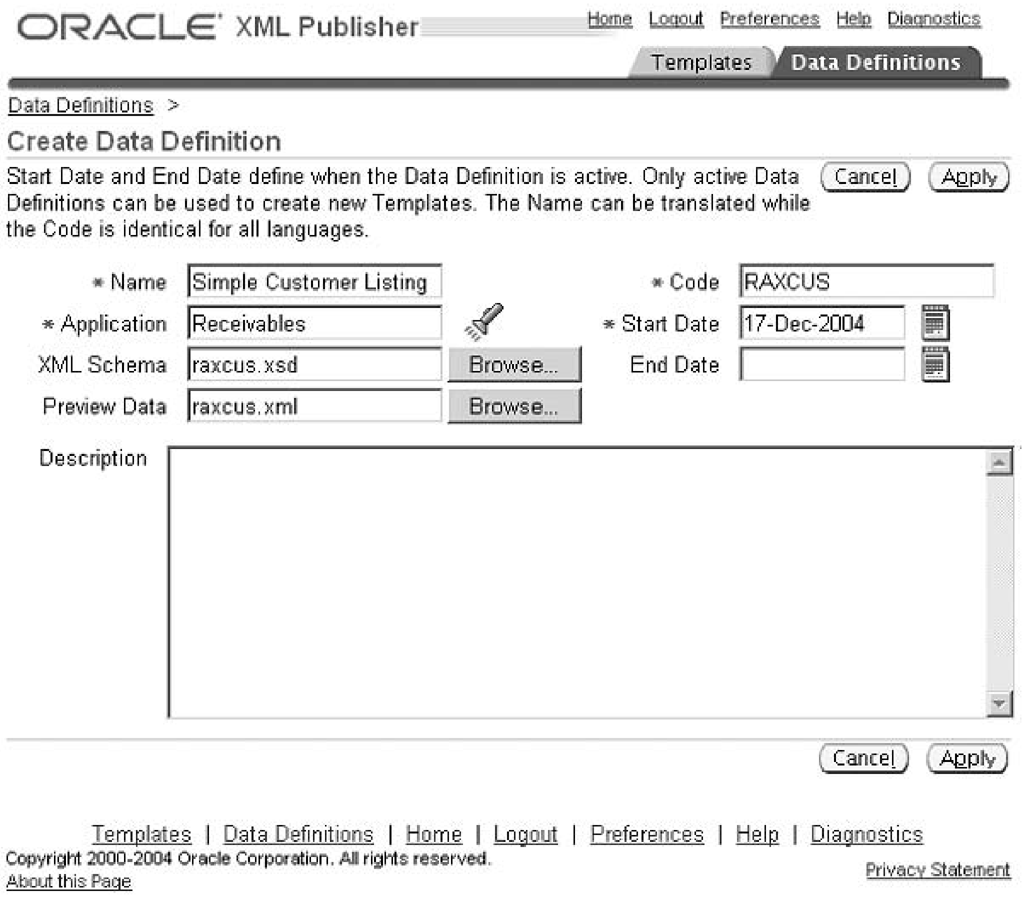
Access the Template Manager from the XML Publisher Administrator responsibility. Select **Templates** to search for or create a template. Select **Data Definitions** to search for or create a data definition.

## Creating the Data Definition

When you create the data definition, you register the source of the data that will be merged with your template layout to create your published report. When you register your template layout file, you must assign it a data definition that exists in the Template Manager. This associates the two at runtime. Multiple templates can use the same data definition.

To navigate to the **Create Data Definition** page:

Select the **Data Definitions** tab, then select the **Create Data Definition** button.


ORACLE XML Publisher
Home Logout Preferences Help Diagnostics
Templates
Data Definitions
Data Definitions >
Create Data Definition

Start Date and End Date define when the Data Definition is active. Only active Data Definitions can be used to create new Templates. The Name can be translated while the Code is identical for all languages.

| | | | |
|---|---|---|---|
| * Name | Simple Customer Listing | * Code | RAXCUS |
| * Application | Receivables | * Start Date | 17-Dec-2004 |
| XML Schema | raxcus.xsd | End Date | |
| Preview Data | raxcus.xml | | |
| Description | | | |

Cancel
Apply
Browse...
Browse...
Templates | Data Definitions | Home | Logout | Preferences | Help | Diagnostics
Copyright 2000-2004 Oracle Corporation. All rights reserved.
About this Page
Privacy Statement

| | |
|---|---|
| **Name** | Enter a user-friendly name for your data definition. |
| **Code** | The data definition **Code** must match the concurrent program short name of the report program (for example, RAXCUS for the Customer Listing Summary). |
| **Application** | Select the report's application from the LOV. |
| **Start Date** | Enter the date from which the data definition will be active. |
| **XML Schema** | You must supply XML Schema if both of the following conditions are applicable: |

- This data definition will be assigned to a PDF template.
- The PDF template will require field mapping.

A PDF template requires mapping if the template form field names (placeholders) do not match the data element tag names of the XML file.

Use the **Browse** button to upload the XML Schema from a saved location.

**Note:** The W3C XML Schema Recommendation defines a standardized language for specifying the structure, content, and certain semantics of a set of XML documents. An XML schema can be considered metadata that describes a class of XML documents. The XML Schema recommendation is described at: http://www.w3.org/TR/xmlschema-0/

For more information, see *Oracle XML DB Developer's Guide 10g*.

| | |
|---|---|
| **End Date** | To make the data definition inactive, enter an end date. |
| **Preview Data** | To use the report **Preview** feature of the Template Manager, upload a sample XML file from the data source. The **Preview** feature is available from the **View Template page**, page 5- 6 . |

After the data definition is created, all the fields are updateable except **Application** and **Code**.

### Viewing and Updating a Data Definition

To view an existing data definition:

1. Search for the data definition from the **Data Definitions** tab.
2. From the search results, select the data definition **Name** to launch the **View Data Definition** page.

Access the **Update Data Definition** page by performing either of the following:

- Select the **Update** icon from the search results region.
- Select the **Update** button from the **View Data Definition** page.

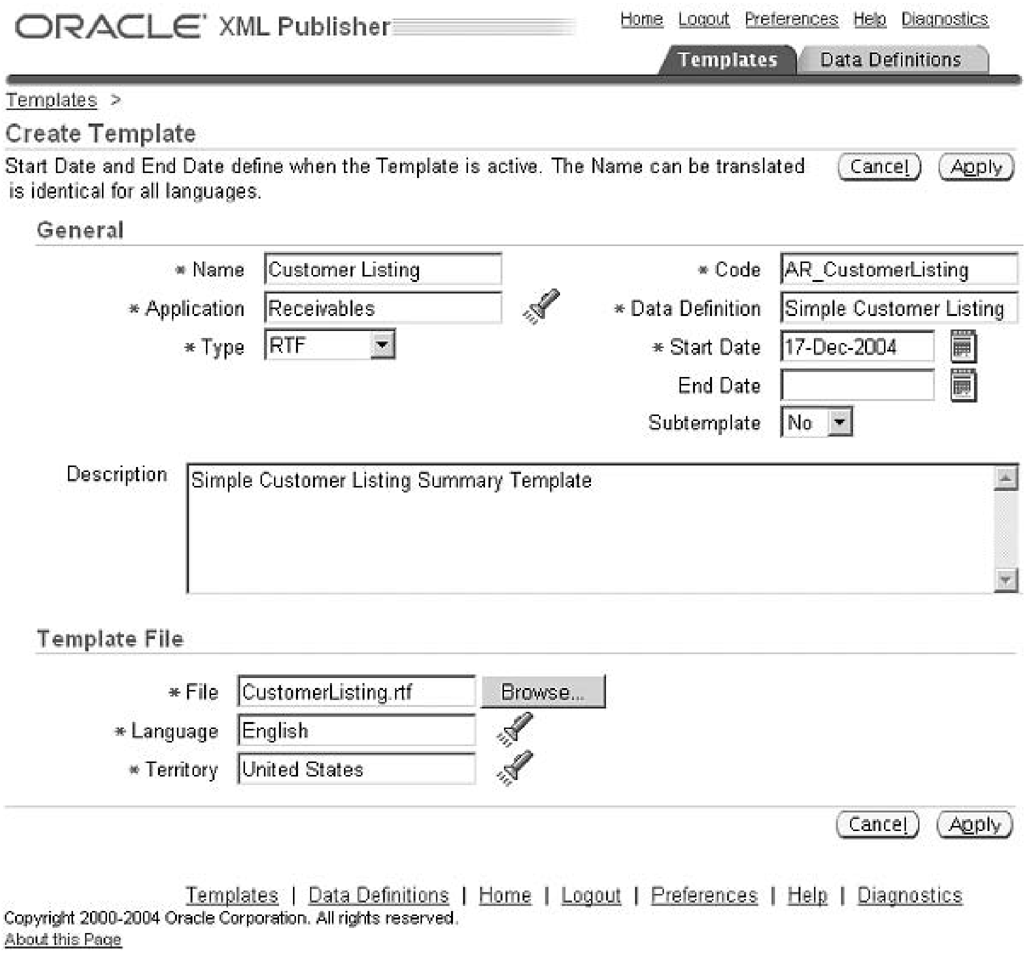
From the **Update Data Definition** page, all fields are updateable except **Application** and **Code**. For information on the updateable fields, see Creating the Data Definition, page 5- 1 .

## Creating the Template

When you create a template, you assign it a data definition and upload your template layout files. Assigning the data definition makes the template available to the corresponding data source at runtime.

At initial creation, you upload one template file for a specific language and territory combination. This file will become the Default Template File (see Default Template File, page 5- 5 ). To upload additional template files or to change the Default Template File, use the **View Template** page (see Viewing and Updating a Template, page 5- 6 ).

If your template type is PDF, the **Template Mapping** region will display after you click the **Apply** button. See Template Mapping, page 5- 5 .



Create Template

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

General

| | | | |
|---|---|---|---|
| * Name | Customer Listing | * Code | AR_CustomerListing |
| * Application | Receivables | * Data Definition | Simple Customer Listing |
| * Type | RTF | * Start Date | 17-Dec-2004 |
| | | End Date | |
| | | Subtemplate | No |

Description: Simple Customer Listing Summary Template

Template File

| | |
|---|---|
| * File | CustomerListing.rtf |
| * Language | English |
| * Territory | United States |

Templates | Data Definitions | Home | Logout | Preferences | Help | Diagnostics
Copyright 2000-2004 Oracle Corporation. All rights reserved.
About this Page

To navigate to the **Create Template** page:

Select the **Templates** tab, then select the **Create Template** button. To copy an existing template, see Copying a Template, page 5- 5 .

| | |
|---|---|
| **Name** | Enter a user-friendly name for your template. |
| **Code** | Assign a template code using the product short name and a descriptive ending. |
| **Application** | Select the report's Application. |
| **Data Definition** | Select your report's data definition. The data definition must already exist in the Template Manager. To register the data definition, see Creating the Data Definition, page 5- 1 . |
| **Type** | Select the file type of the template. Valid template file types are: eText, PDF, RTF, XSL-FO, XSL-HTML, XSL-TEXT, and XSL-XML. |
| **Start Date** | Enter the date from which the template will be active. |
| **End Date** | To make the template inactive, enter an end date. |
| **Subtemplate** | If this is a subtemplate, select "Yes" from the drop list. |

| | |
|---|---|
| | A subtemplate is referenced by other templates, but cannot be run on its own. |
| **File** | Use the **Browse** button to upload your template layout file. |
| **Language** | Select the template language.<br>Add more language template files to your template definition from the **View Template** page. See Adding Templates for Additional Languages, page 5- 7 . |
| **Territory** | Select the language territory. |

After the template definition is created, the following fields are not updateable: **Application**, **Code**, and **Type**. Update the template from the **View Template** page.

## The Default Template

When you submit the XML Publisher concurrent request, you are prompted to specify the language and territory of the template that you wish to apply to the report data. If you do not select the language and territory, XML Publisher will use a template that corresponds to your session language and territory. If your session language and territory combination do not represent an available template, XML Publisher will use the Default Template to publish the report.

When you create the Template definition in the Template Manager, the original template file you upload becomes the Default Template. You can change the Default Template from the **View Template** page by choosing **Update**.

## PDF Template Mapping

If your template type is PDF, the **Template Mapping** region displays after you select **Apply**. If you named the placeholders on the PDF template according to their corresponding XML element names, no mapping is required.

If you did not name the PDF placeholders according to the XML element names (or if you are using a third-party PDF template that already contained named placeholders), you must map each template field name to its corresponding XML element. You must have loaded the XML schema to the template's corresponding Data Definition to make the XML element names available to the Template Manager's mapping tool.

To perform mapping, select the **Enable Mapping** button to launch the **Update Mapping** page. See Mapping PDF Template Fields, page 5- 7 .

For information on creating placeholders in the PDF template, see Creating a Placeholder, page 3- 5 .

## Copying a Template

Use the **Search** region to find the template you wish to copy. From the search results table, select the **Duplicate** icon for the template to launch the **Copy Template** page.

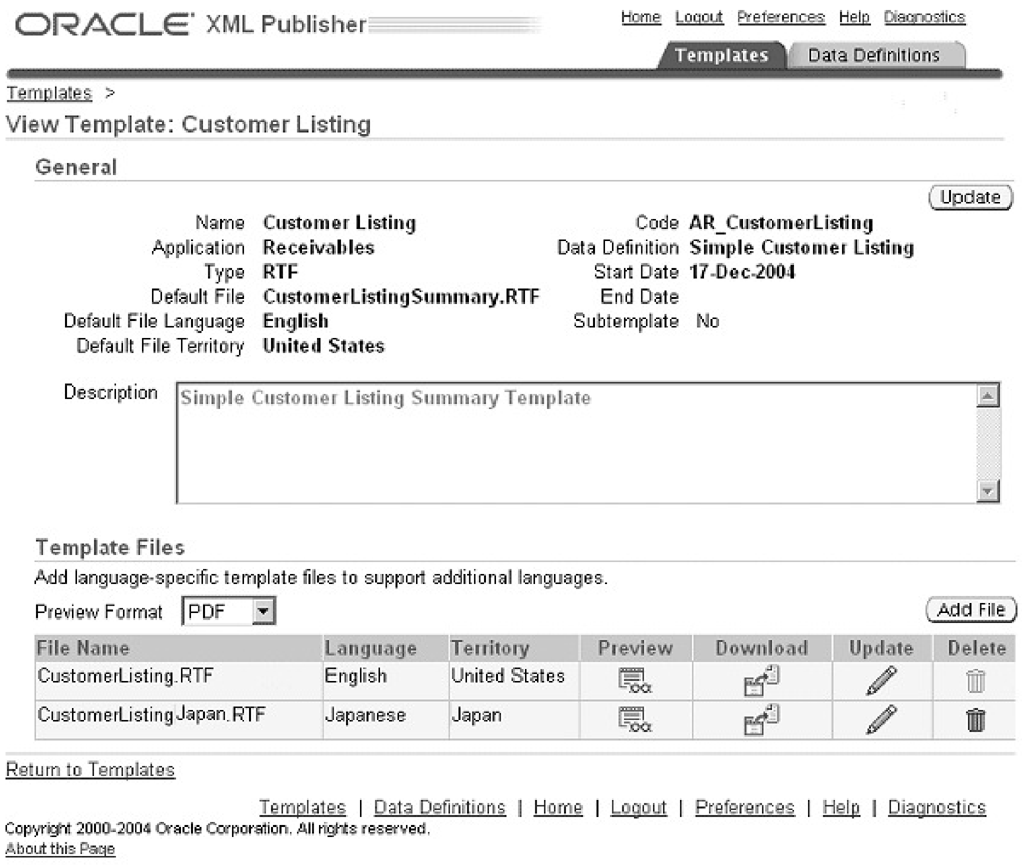
| | |
|---|---|
| **Code** | Assign a template **Code** using the product short name and a descriptive ending. |
| **Name** | Enter a user-friendly name for your template. |
| **Application** | Select the report's application from the LOV. |

| | |
|---|---|
| **Source Template Name** | (Not updateable) Displays the name of the template that you are duplicating. |

## Viewing and Updating a Template

Navigate to the **View Template** page:

1. Search for your template from the **Templates** page.
2. Select the template **Name** from the search results region.


ORACLE XML Publisher
Home Logout Preferences Help Diagnostics
Templates
Data Definitions
Templates >
Update
Add File
Return to Templates
Templates | Data Definitions | Home | Logout | Preferences | Help | Diagnostics
About this Page

View Template: Customer Listing

General

| | | | |
|---|---|---|---|
| Name | Customer Listing | Code | AR_CustomerListing |
| Application | Receivables | Data Definition | Simple Customer Listing |
| Type | RTF | Start Date | 17-Dec-2004 |
| Default File | CustomerListingSummary.RTF | End Date | |
| Default File Language | English | Subtemplate | No |
| Default File Territory | United States | | |

Description: Simple Customer Listing Summary Template

Template Files

Add language-specific template files to support additional languages.

Preview Format: PDF

| File Name | Language | Territory | Preview | Download | Update | Delete |
|---|---|---|---|---|---|---|
| CustomerListing.RTF | English | United States | | | | |
| CustomerListingJapan.RTF | Japanese | Japan | | | | |

Copyright 2000-2004 Oracle Corporation. All rights reserved.

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

- Update the general definitions, page 5- 7
- Preview the template, page 5- 7
- Download the template file, page 5- 7
- Update the template file for editing, page 5- 7
- Add template files for additional languages, page 5- 7
- Update the template field mapping (PDF templates), page 5- 7

> **Note:** Seeded templates cannot be updated or deleted. The Update and Delete icons for these templates are disabled.

## Updating the Template General Definitions

Select the **Update** button to update the general definitions of a template. (You cannot update the **Template Code**, **Template Type**, or **Application**.) For information on the updateable fields, see Creating the Template, page 5- 6 .

## Previewing a Template

If you uploaded a preview data file for your data definition, the **Preview** feature will merge this data file with the selected template to allow you to immediately view a sample of the report within the Template Manager.

Select the **Preview Format** and then select the **Preview** icon next to the template file that you wish to preview. XML Publisher automatically generates a preview of your report in the format selected (PDF templates can only be viewed in PDF format).

## Editing the Template Layout

To edit the layout file of a template:

1. Select the **Download** icon to save the template file to your local file system.
2. Edit the file using your desktop application and save it in the appropriate format.

   For guidelines on creating template files, see Creating an RTF Template, page 2- 1 or Creating a PDF Template, page 3- 1 .
3. Select the **Update** icon.
4. The **Add File** page prompts you to **Browse** for and select your edited file.
5. Select the **Apply** button to upload the edited file to the Template Manager.

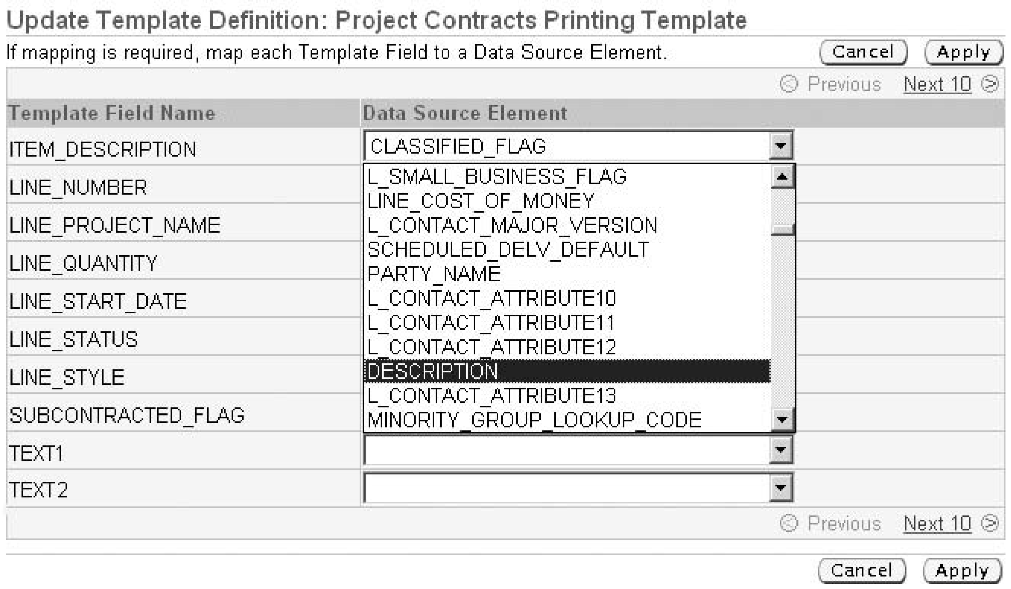
## Adding Templates for Additional Languages

After you have created a template definition, you can add translated template files to support additional languages.

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 | |
| LINE_PROJECT_NAME | |
| LINE_QUANTITY | |
| LINE_START_DATE | |
| LINE_STATUS | |
| LINE_STYLE | |
| SUBCONTRACTED_FLAG | |
| TEXT1 | |
| TEXT2 | |

L_SMALL_BUSINESS_FLAG
LINE_COST_OF_MONEY
L_CONTACT_MAJOR_VERSION
SCHEDULED_DELV_DEFAULT
PARTY_NAME
L_CONTACT_ATTRIBUTE10
L_CONTACT_ATTRIBUTE11
L_CONTACT_ATTRIBUTE12
DESCRIPTION
L_CONTACT_ATTRIBUTE13
MINORITY_GROUP_LOOKUP_CODE

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.

# 6 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

**Important:** Since the latest release of XML Publisher, Applications Object Library (FND) patch 3435480 has been released to fully integrate XML Publisher with the Concurrent Manager. If you have taken this patch, you no longer have to run the XML Publisher Concurrent Request as a second step.

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**.

### If you have applied patch 3435480:

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.

## If you have not applied patch 3435480:

1. Using Standard Request Submission, submit the report, noting the request ID. The request creates the XML data file that XML Publisher will merge with the template.
2. After the request completes, use Standard Request Submission to submit the XML Publisher Concurrent Request.

   The **Parameters** window will prompt you to enter the following fields:

   - Report Request - Select the Request ID of the request you wish to publish.
   - Template - Select the template you wish to use to format the report data. Only templates registered in the Template Manager with the request data source will appear on the list.
   - Template Locale - Select the Language and Territory combination of the template you wish to use.

     **Note:** If you do not select a valid language and territory combination, XML Publisher will use the template that corresponds to your session language and territory. If a valid template for this combination does not exist, XML Publisher will use the Default Template. See Default Template, page 5- 5 .

   - Output Format - select the output format. If your selected template is RTF, you can generate output in Excel (HTML), HTML, PDF, or RTF. If your selected template is PDF, the output format must also be PDF.

When you submit the request, XML Publisher merges the XML data from your chosen request with the selected template to generate your selected output format.

# 7 XML Publisher Extended Functions

## Extended SQL Functions

XML Publisher has extended a set of SQL functions for use in RTF templates. The syntax for these extended functions is

<?xdofx:*expression*?>

The supported functions are shown in the following table:

| SQL Statement | Usage | Description |
|---|---|---|
| 2+3 | <?xdofx:2+3?> | Addition |
| 2-3 | <?xdofx:2-3?> | Subtraction |
| 2*3 | <?xdofx:2*3?> | Multiplication |
| 2/3 | <?xdofx:2/3?> | Division |
| 2**3 | <?xdofx:2**3?> | Exponential |
| 3\|\|2 | <?xdofx:3\|\|2?> | Concatenation |
| lpad('aaa',10,'.') | <?xdofx:lpad('aaa',10,'.')?> | The lpad function pads the left side of a string with a specific set of characters. The syntax for the lpad function is: **`lpad(`**`*string1,padded_ length,[pad_string]*`**`)`** *string1* is the string to pad characters to (the left-hand side). *padded_length* is the number of characters to return. If the padded_length is smaller than the original string, the lpad function will truncate the string to the size of padded_length. *pad_string* is the string that willbe padded to the left-hand side of *string1* . |

| SQL Statement | Usage | Description |
|---|---|---|
| rpad('aaa',10,'.') | <?xdofx:rpad('aaa',10,'.')?> | The rpad function pads the right side of a string with a specific set of characters.<br><br>The syntax for the rpad function is:<br><br>**rpad(***`string1,padded_ length,[pad_string]`***)**.<br><br>*string1* is the string to pad characters to (the right-hand side).<br><br>*padded_length* is the number of characters to return.<br><br>If the padded_length is smaller than the original string, the rpad function will truncate the string to the size of padded_length. *pad_string* is the string that will be padded to the right-hand side of *string1* |
| decode('xxx','bbb','ccc', 'xxx','ddd') | <?xdofx:decode('xxx','bbb', 'ccc','xxx','ddd')?> | The decode function has the functionality of an IF-THEN-ELSE statement. The syntax for the decode function is:<br><br>**`decode(expression, search, result [,search, result]...[, default])`**<br><br>*expression* is the value to compare.<br><br>*search* is the value that is compared against expression.<br><br>*result* is the value returned, if expression is equal to search.<br><br>*default* is returned if no matches are found. |

| SQL Statement | Usage | Description |
|---|---|---|
| Instr('abcabcabc','a',2) | <?xdofx:Instr('abcabcabc', 'a',2)?> | The *instr* function returns the location of a substring in a string. The syntax for the instr function is: **`instr(string1,string2, [start_position],[nth_ appearance])`** *string1* is the string to search. *string2* is the substring to search for in string1. *start_position* is the position in string1 where the search will start. The first position in the string is 1. If the start_position is negative, the function counts back start_position number of characters from the end of string1 and then searches towards the beginning of string1. *nth appearance* is the nth appearance of string2. |
| substr('abcdefg'),2,3) | <?xdofx:substr('abcdefg'),2, 3)?> | The substr function allows you to extract a substring from a string. The syntax for the substr function is: **`substr(string, start_ position, [length])`** *string* is the source string. *start_position* is the position for extraction. The first position in the string is always 1. *length* is the number of characters to extract. |
| replace(name,'John','Jon') | <?xdofx:replace(name, 'John','Jon')?> | The replace function replaces a sequence of characters in a string with another set of characters. The syntax for the replace function is: replace(string1,string_to_replace, [replacement_string]) *string1* is the string to replace a sequence of characters with another set of characters. *string_to_replace* is the string that will be searched for in string1. *replacement_string* is optional. All occurrences of string_to_replace will be replaced with replacement_ string in string1. |
| to_number('12345') | <?xdofx:to_ number('12345')?> | |

| SQL Statement | Usage | Description |
|---|---|---|
| to_char(12345) | <?xdofx:to_char(12345)?> | |
| sysdate() | <?xdofx:sysdate()?> | |

The following table shows supported combination functions:

| SQL Statement | Usage |
|---|---|
| (2+3/4-6*7)/8 | <?xdofx:(2+3/4-6*7)/8?> |
| lpad(substr('1234567890',5,3),10,'^') | <?xdofx:lpad(substr('1234567890',5,3),10,'^')?> |
| decode('a','b','c','d','e','1')\|\|instr('321',1,1) | <?xdofx:decode('a','b','c','d','e','1')\|\| instr('321',1,1)?> |

## XSL Equivalents

The following table lists the XML Publisher simplified syntax with the XSL equivalents.

| Supported XSL Elements | Description | XML Publisher Syntax |
|---|---|---|
| **`<xsl:value-of select= "name">`** | Placeholder syntax | **`<?`***`name`***`?>`** |
| **`<xsl:apply-templates select="name">`** | Applies a template rule to the current element's child nodes. | **`<?apply:`***`name`***`?>`** |
| **`<xsl:copy-of select= "name">`** | Creates a copy of the current node. | **`<?copy-of:`***`name`***`?>`** |
| **`<xsl:call-template name="name">`** | Calls a named template to be inserted into/applied to the current template. | **`<?call:`***`name`***`?>`** |
| **`<xsl:sort select= "name">`** | Sorts a group of data based on an element in the dataset. | **`<?sort:`***`name`***`?>`** |
| **`<xsl:for-each select= "name"`**> | Loops through the rows of data of a group, used to generate tabular output. | **`<?for-each:`***`name`***`?>`** |
| **`<xsl:choose>`** | Used in conjunction with **`when`** and **`otherwise`** to express multiple conditional tests. | **`<?choose?>`** |
| **`<xsl:when test="exp">`** | Used in conjunction with **`choose`** and **`otherwise`** to express multiple conditional tests | **`<?when:`***`expression`***`?>`** |
| **`<xsl:otherwise>`** | Used in conjunction with **`choose`** and **`when`** to express multiple conditional tests | **`<?otherwise?>`** |
| **`<xsl:if test="exp">`** | Used for conditional formatting. | **`<?if:`***`expression`***`?>`** |
| **`<xsl:template name= "name">`** | Template declaration | **`<?template:`***`name`***`?>`** |
| **`<xsl:variable name= "name">`** | Local or global variable declaration | **`<?variable:`***`name`***`?>`** |
| **`<xsl:import href= "url">`** | Import the contents of one stylesheet into another | **`<?import:url?>`** |
| **`<xsl:include href= "url">`** | Include one stylesheet in another | **`<?include:url?>`** |
| **`<xsl:stylesheet xmlns:x="url">`** | Define the root element of a stylesheet | **`<?namespace:x=url?>`** |

## Using FO Elements

You can use any FO element in an RTF template inside the Microsoft Word form fields. The following FO elements have been extended for use with XML Publisher RTF templates. The XML Publisher syntax can be used with either RTF template method.

| FO Element | XML Publisher Syntax |
|---|---|
| `<fo:page-number-citation ref-id= "id">` | `<?fo:page-number-citation:id?>` |
| `<fo:page-number>` | `<?fo:page-number?>` |
| `<fo:ANY NAME WITHOUT ATTRIBUTE>` | `<?fo:ANY NAME WITHOUT ATTRIBUTE?>` |

# 8

# Calling XML Publisher APIs

This chapter covers the following topics:

- Introduction
- PDF Form Processing Engine
- RTF Processor Engine
- FO Processor Engine
- PDF Document Merger
- Document Processor Engine
- XML Publisher Security Properties
- Applications Layer APIs
- Datasource APIs
- Template APIs

## Introduction

This chapter is aimed at developers who wish to create programs or applications that interact with XML Publisher through its application programming interface. This information is meant to be used in conjunction with the Javadocs available from Oracle*MetaLink* document 295036.1, "About Oracle XML Publisher Release 5.0.".

XML Publisher consists of two layers: a core layer of Java APIs and an Applications layer of APIs and UI.

- The core layer contains the main processing engines that parse templates, merge data, generate output, and deliver documents.
- The Applications layer allows the Applications developer to interact with the Template Manager on a programmatic level, which in turn interacts with the core layer.

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

### XML Publisher Core APIs

XML Publisher is made up of the following core API components:

- PDF Form Processing Engine

  Merges a PDF template with XML data (and optional metadata) to produce PDF document output.

- RTF Processor

  Converts an RTF template to XSL in preparation for input to the FO Engine.

- FO Engine

  Merges XSL and XML to produce any of the following output formats: Excel (HTML), PDF, RTF, or HTML.

- PDF Document Merger

  Provides optional postprocessing of PDF files to merge documents, add page numbering, and set watermarks.

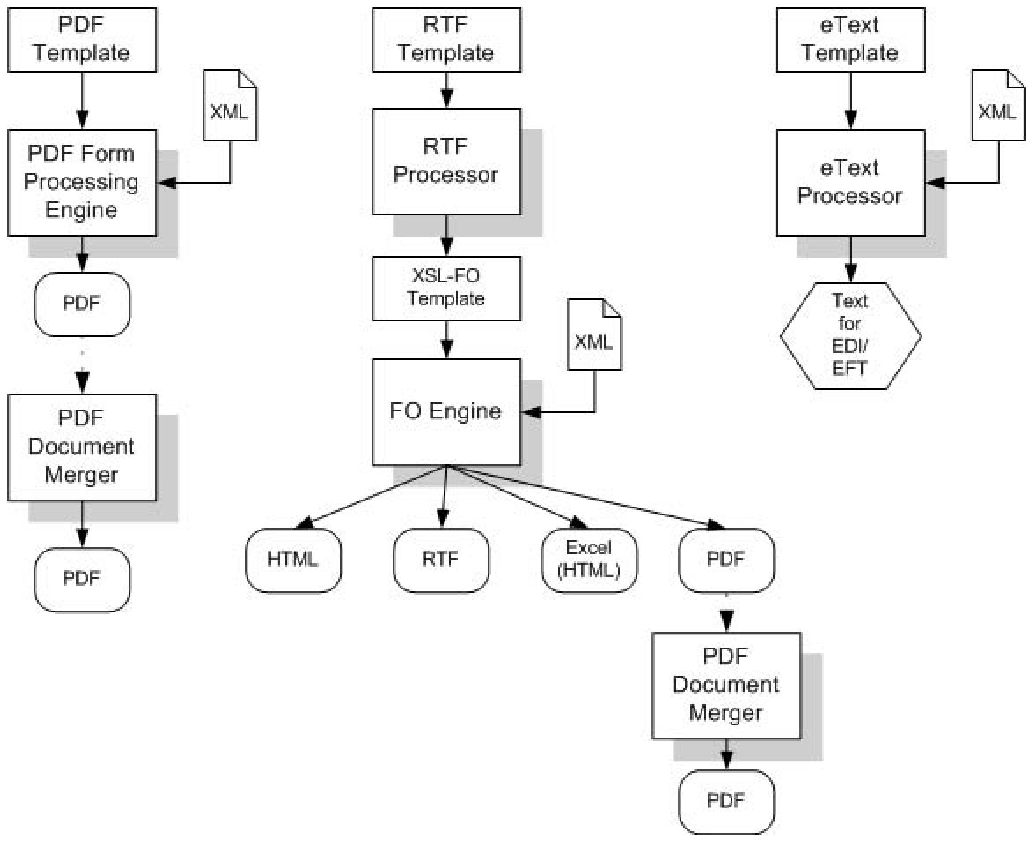
- eText Processor

  Converts RTF eText templates to XSL and merges the XSL with XML to produce text output for EDI and EFT transmissions.

- Document Processor (XML APIs)

  Provides batch processing functionality to access a single API or multiple APIs by passing a single XML file to specify template names, data sources, languages, output type, output names, and destinations.

The following diagram illustrates the template type and output type options for each core processing engine:



## PDF Form Processing Engine

The PDF Form Processing Engine creates a PDF document by merging a PDF template with an XML data file. This can be done using file names, streams, or an XML data string.

As input to the PDF Processing Engine you can optionally include an XML-based Template MetaInfo (.xtm) file. This is a supplemental template to define the placement of overflow data.

The FO Processing Engine also includes utilities to provide information about your PDF template. You can:

- Retrieve a list of field names from a PDF template
- Generate the XFDF data from the PDF template
- Convert XML data into XFDF using XSLT

### Merging a PDF Template with XML Data

XML data can be merged with a PDF template to produce a PDF output document in three ways:

- Using input/output file names
- Using input/output streams
- Using an input XML data string

You can optionally include a metadata XML file to describe the placement of overflow data in your template.

#### Merging XML Data with a PDF Template Using Input/Output File Names

Input:

- Template file name (String)
- XML file name (String)
- Metadata XML file name (String)

Output:

- PDF file name (String)

**Example**

```
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); // Inp
ut Data
  FileInputStream fIs3 = new FileInputStream(metaData); // Metadat
a XML Data
  FileOutputStream fOs = new FileOutputStream(newFilePath); // Out
put 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 : " + fProce
ssor.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/X
SL/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 v
alue 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 a
s a text of the "value" element. -->
            <xsl:value-of select="." />
         </value>
      </field>
   </xsl:template>
</xsl:stylesheet>
```

You can then use the XFDFObject to convert XML to the XFDF format using an XSLT as follows:

**Example**

```
import java.io.*;
import oracle.apps.xdo.template.pdf.xfdf.XFDFObject;
.
.
.
XFDFObject xfdfObject  = new XFDFObject();

xfdfObject .setStylesheet(new BufferedInputStream(new FileInputStr
eam(xslPath)));   // XSL file name
xfdfObject .appendXML( new File(xmlPath1));   // XML data file nam
e
xfdfObject .appendXML( new File(xmlPath2));   // XML data file nam
e

System.out.print(xfdfObject .toString());
```

## RTF Processor Engine

### Generating XSL

The RTF processor engine takes an RTF template as input. The processor parses the template and creates an XSL-FO template. This can then be passed along with a data source (XML file) to the FO Engine to produce PDF, HTML, RTF, or Excel (HTML) output.

Use either input/output file names or input/output streams as shown in the following examples:

#### Generating XSL with Input/Output File Names

Input:

- RTF file name (String)

Output:

- XSL file name (String)

**Example**

```
import oracle.apps.xdo.template.FOProcessor;
.
.
.
  public static void main(String[] args)
  {
RTFProcessor rtfProcessor = new RTFProcessor(args[0]); //input tem
plate
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]);  //inpu
t template
    FileOutputStream  fOs  = new FileOutputStream(args[1]); // out
put

    RTFProcessor rtfProcessor = new RTFProcessor(fIs);
    rtfProcessor.setOutput(fOs);
    rtfProcessor.process();
    // Closes inputStreams 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. The FO Processor can generate output in PDF, RTF, HTML, or Excel (HTML) from either of the following two inputs:

- Template (XSL) and Data (XML) combination
- FO object

Both input types can be passed as file names, streams, or in an array. Set the output format by setting the setOutputFormat method to one of the following:

- FORMAT_EXCEL
- FORMAT_HTML
- FORMAT_PDF
- FORMAT_RTF

An XSL-FO utility is also provided that creates XSL-FO from the following inputs:

- XSL file and XML file
- Two XML files and two XSL files
- Two XSL-FO files (merge)

The FO object output from the XSL-FO utility can then be used as input to the FO processor.

## Major Features of the FO Processor

### Bidirectional Text

XML Publisher utilizes the Unicode BiDi algorithm for BiDi layout. Based on specific values for the properties writing-mode, direction, and unicode bidi, the FO Processor supports the BiDi layout.

The writing-mode property defines how word order is supported in lines and order of lines in text. That is: right-to-left, top-to-bottom or left-to-right, top-to-bottom. The direction property determines how a string of text will be written: that is, in a specific direction, such as right-to-left or left-to-right. The unicode bidi controls and manages override behavior.

### Font Fallback Mechanism

The FO Processor supports a two-level font fallback mechanism. This mechanism provides control over what default fonts to use when a specified font or glyph is not found. XML Publisher provides appropriate default fallback fonts automatically without requiring any configuration. XML Publisher also supports user-defined configuration files that specify the default fonts to use. For glyph fallback, the default mechanism will only replace the glyph and not the entire string.

For more information, see XML Publisher Configuration File, page A- 1 .

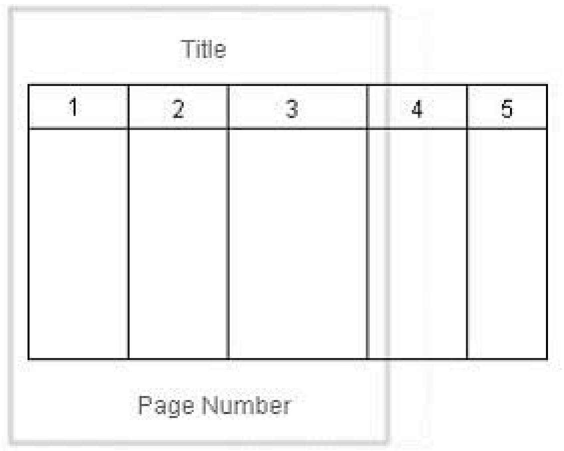
### Variable Header and Footer

For headers and footers that require more space than what is defined in the template, the FO Processor extends the regions and reduces the body region by the difference between the value of the page header and footer and the value of the body region margin.

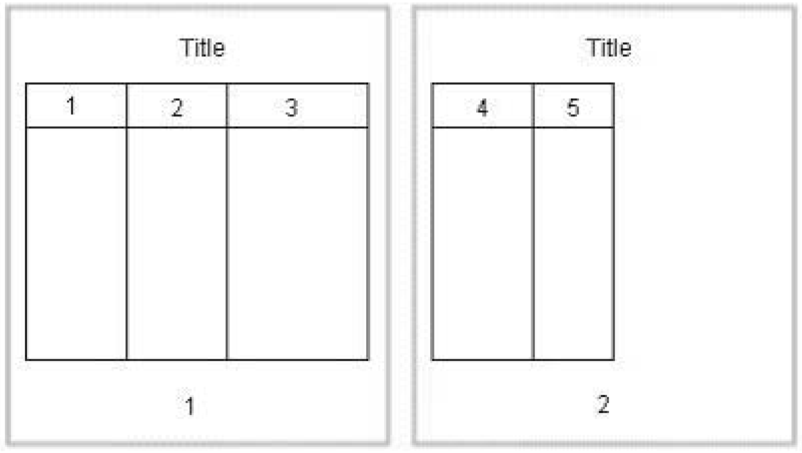
### Horizontal Table Break

This feature supports a "Z style" of horizontal table break. The horizontal table break is not sensitive to column span, so that if the column-spanned cells exceed the page (or area width), the FO Processor splits it and does not apply any intelligent formatting to the split cell.

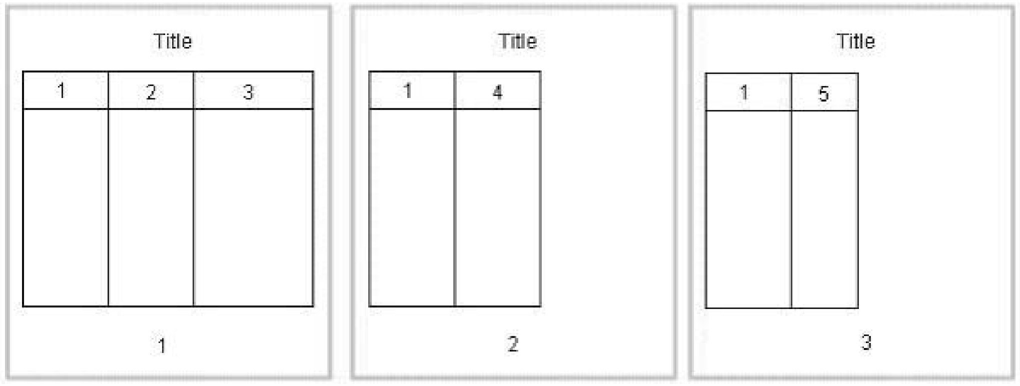
The following figure shows a table that is too wide to display on one page:



The following figure shows one option of how the horizontal table break will handle the wide table. In this example, a horizontal table break is inserted after the third column.



The following figure shows another option. The table breaks after the third column, but includes the first column with each new page.



### Generating Output Using File Names

The following example shows how to use the FO Processor to create an output file using file names.

Input:

- XML file name (String)
- XSL file name (String)

Output:

- Output file name (String)

**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) ou
tput file
    processor.setOutputFormat(FOProcessor.FORMAT_PDF); processor.p
rocess();
    // Start processing
    try
    {
       processor.generate();
    }
    catch (XDOException e)
    {
       e.printStackTrace();
       System.exit(1);
    }

  }
```

## Using the XSL-FO Utility

Use the XSL-FO Utility to create an XSL-FO output file from input XML and XSL files, or to merge two XSL-FO files. Output from this utility can be used to generate your final output. See Generating Output from an XSL-FO file, page 8-17.

### 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 xs
l-fo input

    FOProcessor processor = new FOProcessor();
    processor.setData(xmlInput);
    processor.setTemplate(xslInput);

    processor.setOutput("/tmp/output.pdf);          //set (PDF) ou
tput file
    processor.setOutputFormat(FOProcessor.FORMAT_PDF); processor.p
rocess();
    // 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[inputNu
mbers];
      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, outp
utStream);

      // 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, OutputStrea
m outputStream)
  {
    try
    {
      // Initialize PDFDocMerger
      PDFDocMerger docMerger = new PDFDocMerger(inputStreams, outp
utStream);

      // 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, Input
Stream backgroundStream, OutputStream outputStream)

 {
    try
    {
      // Initialize PDFDocMerger
      PDFDocMerger docMerger = new PDFDocMerger(inputStreams, outp
utStream);

      // 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 Arrary)

Output:

- PDF Document (OutputStream)

**Example**

```
import java.io.*;
import oracle.apps.xdo.common.pdf.util.PDFDocMerger;
.
.
.
  public boolean mergeDocs(InputStream[] inputStreams, OutputStrea
m outputStream)
  {
    try
    {
      // Initialize PDFDocMerger
      PDFDocMerger docMerger = new PDFDocMerger(inputStreams, outp
utStream);

      // 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"(Helv
etica, 8)" is used.
      docMerger.setPageNumberFontInfo("Courier", 8);

      // If this method is not called, then the default initial va
lue "(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 starting 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, outp
utStream);

      // You can use setTextDefaultWatermark() without these detai
led setting
      docMerger.setTextWatermark("DRAFT", 200f, 200f); //set text
and place
      docMerger.setTextWatermarkAngle(80);                //set an
gle
      docMerger.setTextWatermarkColor(1.0f, 0.3f ,0.5f);  // set R
GB 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 imageFilePath)
  {
    try
    {
      // Initialize PDFDocMerger
      PDFDocMerger docMerger = new PDFDocMerger(inputStreams, outp
utStream);

      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;
    }
  }
```

## 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, 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

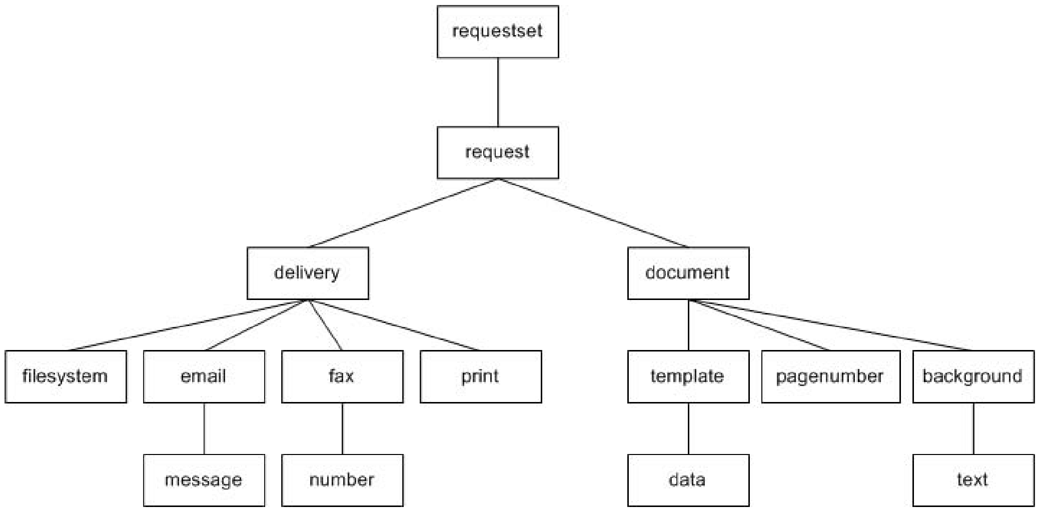
- Provides sample XML files to demonstrate specific processing options
- Provides example code to invoke the processors

## 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`**<br>**`version`** | Root element must contain [**`xmlns:xapi="http://xmlns.oracle.com/oxp/xapi/"`**] block<br>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:<br>pdf (Default if not specified)<br>rtf<br>html<br>excel<br>text<br>. |
| **filesystem** | **output** | Specify this element to save the output to the file system. Define the directory path in the **output** attribute. |
| **print** | • **printer**<br>• **server-alias** | 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** | • **server**<br>• **server-alias** | 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. |
| **email** | • **server**<br>• **port**<br>• **from**<br>• **reply-to**<br>• **server-alias** | Specify the outgoing mail server (SMTP) in the **server** attribute.<br>Specify the mail server port in the **port** attribute. |

| Element | Attributes | Description |
|---|---|---|
| **`message`** | • **`to`**<br>• **`cc`**<br>• **`bcc`**<br>• **`attachment`**<br>• **`subject`** | The **`message`** element can be placed several times under the **`email`** element. You can specify character data in the **`message`** element.<br>You can specify multiple email addresses in the **`to`**, **`cc`** and **`bcc`** attributes separated by a comma.<br>The **`attachment`** value is either true or false (default). If **`attachment`** is true then a generated document will be attached when the email is sent.<br>The **`subject`** attribute is optional. |
| **`background`** | **`where`** | If the background text is required on a specific page, then set the **`where`** value to the page numbers required. The page index starts at 1. The default value is 0, which places the background on all pages. |
| **`text`** | • **`title`**<br>• **`default`** | Specify the watermark text in the **`title`** value.<br>A **`default`** value of "yes" automatically draws the watermark with forward slash type. The default value is yes. |
| **`pagenumber`** | • **`initial-page-index`**<br>• **`initial-value`**<br>• **`x-pos`**<br>• **`y-pos`** | The **`initial-page-index`** default value is 0.<br>The **`initial-value`** default value is 1.<br>"Helvetica" is used for the page number font.<br>The **`x-pos`** provides lower left x position.<br>The **`y-pos`** provides lower left y position. |

| Element | Attributes | Description |
|---|---|---|
| **`template`** | • **`locale`**<br>• **`location`**<br>• **`type`** | Contains template information.<br>Valid values for the **`type`** attribute are<br>pdf<br>rtf<br>xsl-fo<br>etext<br>The default value is "pdf". |
| **`data`** | **`location`** | Define the **`location`** attribute to specify the location of the data, or attach the actual XML data with subelements. The default value of **`location`** is "inline". It the **`location`** points to either an XML file or a URL, then the data should contain an XML declaration with the proper encoding.<br>If the **`location`** attribute is not specified, the **`data`** element should contain the subelements for the actual data. This must not include an XML declaration. |

## XML File Samples

Following are sample XML files that show:

- Simple XML shape
- Defining two data sets
- Defining multiple templates and data
- Retrieving templates over HTTP
- Retrieving data over HTTP
- Generating more than one output
- Defining page numbers

### Simple XML sample

The following sample is a simple example that shows the definition of one template (**`template1.pdf`**) and one data source (**`data1`**) to produce one output file (**`outfile.pdf`**) delivered to the file system:

**Example**

```
<?xml version="1.0" encoding="UTF-8" ?>
     <xapi:requestset xmlns:xapi="http://xmlns.oracle.com/oxp/xapi
">
       <xapi:request>
         <xapi:delivery>
           <xapi:filesystem output="d:\tmp\outfile.pdf" />
         </xapi:delivery>
         <xapi:document output-type="pdf">
           <xapi:template type="pdf" location="d:\mywork\template1
.pdf">
             <xapi:data>
               <field1>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/temp
late1.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/temp
late2.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/temp
late1.pdf">
        <xapi:data location="http://your.server:9999/data/data_1.x
ml"/>
        <xapi:data location="http://your.server:9999/data/data_2.x
ml"/>
      </xapi:template>

      <xapi:template type="pdf"
                  location="http://your.server:9999/templates/temp
late2.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(data
File, tempDir);
      docProcessor.process();
    }
    catch(Exception e)
    {
 e.printStackTrace();
       System.exit(1);
    }
    System.exit(0);
  }
```

### Invoke Processors with InputStream

Input:

- Data file (InputStream)
- Directory for Temporary Files (String)

**Example**

```
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);
  }
```

## XML Publisher Security 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 Security Properties, page A-10 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

### Passing Properties to the FO Engine

To pass a property as a Property object, set the name/value pair for the property prior to calling the FO Processor, as shown in the following example:

Input:

- XML file name (String)
- XSL file name (String)

Output:

- PDF file name (String)

**Example**

```
import oracle.apps.xdo.template.FOProcessor;
.
.
.
  public static void main(String[] args)
  {

    FOProcessor processor = new FOProcessor();
    processor.setData(args[0]);     // set XML input file
    processor.setTemplate(args[1]); // set XSL input file
    processor.setOutput(args[2]);  //set (PDF) output file
    processor.setOutputFormat(FOProcessor.FORMAT_PDF);
    Properties prop = new Properties();
    prop.put("pdf-security", "true");             // PDF security
control
    prop.put("pdf-permissions-password", "abc");  // permissions p
assword
    prop.put("pdf-encription-level", "0");        // encryption le
vel
    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        p
rocessor.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**

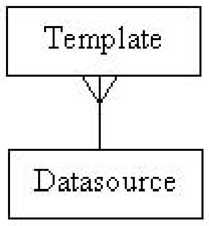
```
import oracle.apps.xdo.batch.DocumentProcessor;
.
.
.
  public static void main(String[] args)
  {
.
.
.
    try
    {
      // dataFile --- File path of the Document Processor XML
      // tempDir  --- Temporary Directory path
      DocumentProcessor  docProcessor = new DocumentProcessor(data
File, tempDir);
      Properties prop = new Properties();
      prop.put("pdf-security", "true");             // PDF securit
y control
      prop.put("pdf-permissions-password", "abc");  // permissions
 password
      prop.put("pdf-encription-level", "0");        // encryption
level
      processor.setConfig(prop);
      docProcessor.process();
    }
    catch(Exception e)
    {
 e.printStackTrace();
       System.exit(1);
    }
    System.exit(0);
  }
```

## Applications Layer APIs

The applications layer of XML Publisher allows you to store and manager data sources and templates through the Template Manager user interface via the XML Publisher Administrator responsibility. You can also access and manipulate these objects via an application programmer's interface. 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 defintion 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 OAApplicationModule.

2. If you are using this class in a Java concurrent program, pass CpContext as an AppsContext.

3. Otherwise create AppsContext from the DBC file. If you are running a servlet/JSP in Applications, you can obtain the full path to the DBC file by calling

   **`System.getProperty("JTFDBCFILE")`** or **`System.getProperty("BNEDBCFILE"`**)

### 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", "TestDataSou
rce");
 // 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.

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 OAApplicationModule.

2. If you are using this class in a Java concurrent program, pass CpContext as an AppsContext.

3. Otherwise create AppsContext from the DBC file. If you are running a servlet/JSP in Applications, you can obtain the full path to the DBC file by calling

   **`System.getProperty("JTFDBCFILE"`**) or **`System.getProperty("BNEDBCFILE"`**)

### Getting the OAApplicationModule Instance

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

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

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

It is recommended that you use AppsContext to call APIs because the latest development is based on the APIs that take AppsContext. You can still use APIs that take OAApplicationModule, but they internally call corresponding APIs that take AppsContext.

### Creating Template Entries

To add a new template entry to the Template Manager repository:

1. Create an instance of the Template class by calling the **`Template.createInstance()`** method

2. Set the attributes of the instance.

3. Pass it to the **`TemplateHelper.createTemplate()`** method

**Example**

```
// Create an instance
 Template t = Template.createInstance(appsContext, "XDO", "TestTem
plate",
   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", "X
DO%", true);

 // Update properties
 t[0].setDescription("updated template entry.");
 t[0].setStartDate(new java.sql.Date(System.currentTimeMillis()));
 t[0].setName("updated template entry name");
 t[0].setStatus(TypeDefinitions.TEMPLATE_STATUS_ENABLED);

 // Call updateTemplate() to commit the update in the repository
 TemplateHelper.updateTemplate(appsContext, t[0]);
```

### Deleting Template Entries

Delete template entries by calling the **Template.deleteTemplate**() method. The method does not actually delete the record from the repository, but marks it as "disabled" for future use. You can change the status anytime by calling the **Template.updateTemplateStatus()** method.

### Adding, Updating, and Deleting Template Files

You can add, update and delete template files by calling methods defined in the TemplateHelper class. Please note that unlike the template entries, deleting template files actually deletes the record from the repository.

The following code sample demonstrates adding, deleting, and updating a template file:

**Example**

```
// Add English template file to the template entry
TemplateHelper.addTemplateFile(
 appsContext,           // AppsContext
 "XDO",                 // Application short name of the template
 "TestTemplate",        // Template code of the template
 "en",                  // ISO language code of the template
 "US",                  // ISO territory code of the template
 Template.TEMPLATE_TYPE_PDF, // Type of the template file
 "us.pdf",              // Filename of the template file
 new FileInputStream("/path/to/us.pdf")); // Template file

// Add Japanese template file to the template entry
TemplateHelper.addTemplateFile(
 appsContext,           // AppsContext
 "XDO",                 // Application short name of the template
 "TestTemplate",        // Template code of the template
 "ja",                  // ISO language code of the template
 "JP",                  // ISO territory code of the template
 Template.TEMPLATE_TYPE_PDF, // Type of the template file
 "ja.pdf",              // Filename of the template file
 new FileInputStream("/path/to/ja.pdf")); // Template file

// Update English template file to the template entry
TemplateHelper.updateTemplateFile(
 appsContext,           // AppsContext
 "XDO",                 // Application short name of the template
 "TestTemplate",        // Template code of the template
 "en",                  // ISO language code of the template
 "US",                  // ISO territory code of the template
 Template.TEMPLATE_TYPE_PDF, // Type of the template file
 "us.pdf",              // Filename of the template file
 new FileInputStream("/path/to/new/us.pdf")); // Template file

// Delete Japanese template file to the template entry
TemplateHelper.deleteTemplateFile(
 appsContext,           // AppsContext
 "XDO",                 // Application short name of the template
 "TestTemplate",        // Template code of the template
 "ja",                  // ISO language code of the template
 "JP");                 // ISO territory code of the template
```

### Getting Template Files

Download template file contents from the repository by calling the `getTemplateFile()` methods. These methods return an InputStream connected to the template file as a return value.

**Example**

```
// Download the English template file from the repository
 InputStream in = TemplateHelper.getTemplateFile(
  appsContext,           // AppsContext
  "XDO",                 // Application short name of the template
  "TestTemplate",        // Template code of the template
  "en",                  // ISO language code of the template
  "US");                 // ISO territory code of the template
```

## Processing Templates

You can apply a template, stored in the Template Manager, to an XML data source by calling one of the processTemplate() methods. You need to pass the OutputStream object for the destination of the processed document.

### Example

```
// 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 procesed
document
  properties,            // Properties for the template processing
  docOutputStream)       // OutputStream where the processed docum
ent goes.
```

Pass the properties for template processing by passing a Properties object. You can pass **`null`** if you have nothing to tell to the XML Publisher processors.

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 procese
d document
  properties,            // Properties for the template processing
.
                         // All properties will be passed to EFT/E
DI engine
  docOutputStream)       // OutputStream where the processed docum
ent 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 5- 5 for more information on assigning a default template file.

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

| **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.

# 9 Delivery Manager

## 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, or HTTP) or custom delivery channels
- Track the status of each delivery
- Redeliver documents

### Using the Delivery Manager

To use the Delivery Manager follow these steps:

1. Create a DeliveryManager instance
2. Create a DeliveryRequest instance using the createRequest() method
3. Add the request properties (such as DeliveryRequest destination). Most properties require a String value. See the supported properties for each delivery channel for more information.
4. Set your document to the DeliveryRequest.
5. Call submit() to submit the delivery request.

One delivery request can handle one document and one destination. This facilitates monitoring and resubmission, if necessary.

DeliveryRequest allows you to set the documents in three ways as follows:

- Get OutputStream from the DeliveryRequest and write the document to the OutputStream. You do not need to close the OutputStream to call the submit() method immediately after you finish writing the document to the OutputStream.
- Set InputStream of the document to DeliveryRequest. The DeliveryRequest will read the InputStream when you call submit() for the first time. The DeliveryRequest does not close the InputStream so you must ensure to close it.
- Set the file name of the document to DeliveryRequest.

The Delivery Manager supports streamlined delivery when you set the direct mode. See Direct and Buffering Modes, page 9-17.

The follow delivery channels are described in this document:

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

## 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_S
MTP_EMAIL);

     // set email subject
     req.addProperty(DeliveryPropertyDefinitions.SMTP_SUBJECT, "te
st mail");
     // set SMTP server host
     req.addProperty(
       DeliveryPropertyDefinitions.SMTP_HOST, "mysmtphost");
     // set the sender email address
     req.addProperty(DeliveryPropertyDefinitions.SMTP_FROM, "mynam
e@mydomain.com");
     // set the destination email address
     req.addProperty(
       DeliveryPropertyDefinitions.SMTP_TO_RECIPIENTS, "user1@mydo
main.com, user2@mydomain.com" );
     // set the content type of the email body
     req.addProperty(DeliveryPropertyDefinitions.SMTP_CONTENT_TYPE
, "application/pdf");
     // set the document file name appeared in the email
     req.addProperty(DeliveryPropertyDefinitions.SMTP_CONTENT_FILE
NAME, "test.pdf");
     // set the document to deliver
     req.setDocument("/document/test.pdf");

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

The following table lists the supported properties:

| Property | Description |
|---|---|
| SMTP_TO_RECIPIENTS | Required<br>Enter multiple recipients separated by a comma (example: "user1@mydomain.com, user2@mydomain.com") |
| SMTP_CC_RECIPIENTS | Optional<br>Enter multiple recipients separated by a comma. |
| SMTP_BCC_RECIPIENTS | Optional<br>Enter multiple recipients separated by a comma. |
| SMTP_FROM | Required<br>Enter the e-mail address of the sending party. |
| SMTP_REPLY_TO | Optional<br>Enter the reply-to e-mail address. |
| SMTP_SUBJECT | Required<br>Enter the subject of the e-mail. |
| SMTP_CHARACTER_ENCODING | Optional<br>Default is "UTF-8". |
| SMTP_ATTACHMENT | Optional<br>If you are including an attachment, enter the attachment object name. |
| SMTP_CONTENT_FILENAME | Required<br>Enter the file name of the document (example: invoice.pdf) |
| SMTP_CONTENT_TYPE | Required<br>Enter the MIME type. |
| SMTP_SMTP_HOST | Required<br>Enter the SMTP host name. |
| SMTP_SMTP_PORT | Optional<br>Enter the SMTP port. Default is 25. |
| SMTP_SMTP_USERNAME | Optional<br>If the SMTP server requires authentication, enter your username for the server. |

| Property | Description |
|---|---|
| SMTP_SMTP_PASSWORD | Optional<br>If the SMTP server requires authentication, enter the password for the username you entered. |
| SMTP_ATTACHMENT_FIRST | Optional<br>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.c
om");

     // set the CC email addresses
     req.addProperty(
       DeliveryPropertyDefinitions.SMTP_CC_RECIPIENTS,
        "user4@mydomain.com, user5@mydomain.com, user6@mydomain.co
m");
```

### Attaching Multiple Documents into One Request

Use the Attachment utility class to attach multiple documents into one request. Sample usage is as follows:

**Example**

```
// Properties for Attachment
   Hashtable props = new Hashtable();
   // Set encoding property for the non-ASCII file names.
   // It's optional. Default value is "UTF-8"
   props.put(DeliveryPropertyDefinitions.SMTP_CHARACTER_ENCODING,
"UTF-8");
     :
     :
   (You can append other properties also)
     :
     :

   // create Attachment instance
   Attachment m = new Attachment(props);

   // add attachment files
   m.addAttachment(
       "/pdf_sample/pdfTest5.pdf",        // source file name
       "a1.pdf",                          // file name appeared on
 the email
       "application/pdf");                // content type
   m.addAttachment(
       "/rtf_sample/rtfsample_en00.rtf",  // source file name
       "a2.rtf",                          // file name appeared on
 the email
       "application/rtf");                // content type
   m.addAttachment(
       "/xml_sample/pdfTest5.xml",        // source file name
       "a3.xml",                          // file name appeared on
 the email
       "text/xml");                       // content type

     :
     :

   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_FIR
ST, "true");
     :
```

### Providing Username and Password for Authentication

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

**Example**

```
 :
   req.addProperty(DeliveryPropertyDefinitions.SMTP_USERNAME, "sco
tt");
   req.addProperty(DeliveryPropertyDefinitions.SMTP_PASSWORD, "tig
er");
     :
```

## Delivering Your Document to a Printer

The Delivery Server supports Internet Printing Protocol (IPP) as defined in RFC 2910 and 2911 for the delivery of documents to IPP-supported printers or servers, such as CUPS.

Common Unix Printing System (CUPS) is a free, server-style, IPP-based software that can accept IPP requests and dispatch those requests to both IPP and non-IPP based devices, such as printers and fax machines. See http://www.cups.org/ for more details.

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_I
PP_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_FORM
AT,
       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<br>Enter the host name. |
| IPP_PORT | Optional<br>Default is 631. |
| IPP_PRINTER_NAME | Required<br>Enter the name of the printer that is to receive the output (example: /printers/myPrinter). |
| IPP_AUTHTYPE | Optional<br>Valid values for authentication type are:<br>IPP_AUTHTYPE_NONE - no authentication (default)<br>IPP_AUTHTYPE_BASIC - use HTTP basic authentication<br>IPP_AUTHTYPE_DIGEST - use HTTP digest authentication |

| Property | Description |
|---|---|
| IPP_USERNAME | Optional<br>Enter the username for HTTP authentication. |
| IPP_PASSWORD | Optional<br>Enter the password for HTTP authentication. |
| IPP_ENCTYPE | Optional<br>The encryption type can be set to either of the following:<br>IPP_ENCTYPE_NONE - no encryption (default)<br>IPP_ENCTYPE_SSL - use Secure Socket Layer |
| IPP_USE_FULL_URL | Optional<br>Set to "true" to send the full URL for the HTTP request header.  Valid values are "true" or "false" (default). |
| IPP_USE_CHUNKED_BODY | Optional<br>Valid values are "true" (default) to use HTTP chunked transfer coding for the message body, or "false". |
| IPP_ATTRIBUTE_CHARSET | Optional<br>Attribute character set of the IPP request. Default is "UTF-8". |
| IPP_NATURAL_LANGUAGE | Optional<br>The natural language of the IPP request. Default is "en". |
| IPP_JOB_NAME | Optional<br>Job name of the IPP request. |
| IPP_COPIES | Optional<br>Define the number of copies to print (example: "1" , "5", "10"). Default is 1. |
| IPP_SIDES | Optional<br>Enable two-sided printing. This setting will be ignored if the target printer does not support two-sided printing. Valid values are:<br>IPP_SIDES_ONE_SIDED - default<br>IPP_SIDES_TWO_SIDED_LONG_EDGE - prints both sides of paper for binding long edge.<br>IPP_SIDES_TWO_SIDED_SHORT_EDGE - prints both sides of paper for binding short edge. |

| Property | Description |
|---|---|
| IPP_ORIENTATIONS | Optional<br>Sets the paper orientation. This setting will be ignored if the target printer does not support orientation settings. Valid values are:<br>IPP_ORIENTATIONS_PORTRAIT (default)<br>IPP_ORIENTATIONS_LANDSCAPE |
| IPP_DOCUMENT_FORMAT | Optional<br>The target printer must support the specified format. Valid values are:<br>IPP_DOCUMENT_FORMAT_POSTSCRIPT<br>IPP_DOCUMENT_FORMAT_PLAINTEXT<br>IPP_DOCUMENT_FORMAT_PDF<br>IPP_DOCUMENT_FORMAT_ OCTETSTREAM (default) |

### Printing over an HTTP Proxy Server

To deliver documents to IPP printers or fax machines over an HTTP proxy server, you may encounter delivery problems due to differences in the HTTP implementations between CUPS and the proxy servers. Setting the following two properties can resolve most of these problems:

- DeliveryPropertyDefinitions.IPP_USE_FULL_URL - set to "true"
- DeliveryPropertyDefinitions.IPP_USE_CHUNKED_BODY - set to "false"

## Delivering Your Documents via Fax

The delivery system supports the delivery of documents to fax modems configured on CUPS. You can configure fax modems on CUPS with efax (http://www.cce.com/efax/ [(http://www.cce.com/efax/]) and FAX4CUPS (http://gongolo.usr.dsi.unimi.it/fax4CUPS/).

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_I
PP_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_FORM
AT, "application/postscript");
     // set the phone number to send
     req.addProperty(DeliveryPropertyDefinitions.IPP_PHONE_NUMBER,
 "9999999");
     // set the document
     req.setDocument("/document/invoice.pdf");

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

The supported properties are the same as those supported for printer documents, plus the following:

| Property | Description |
|---|---|
| IPP_PHONE_NUMBER | Required<br>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_W
EBDAV);

     // set document content type
     req.addProperty(DeliveryPropertyDefinitions.WEBDAV_CONTENT_TY
PE, "application/pdf");
     // set the WebDAV server hostname
     req.addProperty(DeliveryPropertyDefinitions.WEBDAV_HOST, "myw
ebdavhost");
     // set the WebDAV server port number
     req.addProperty(DeliveryPropertyDefinitions.WEBDAV_PORT, "80"
);
     // set the target remote directory
     req.addProperty(DeliveryPropertyDefinitions.WEBDAV_REMOTE_DIR
ECTORY, "/content/");
     // set the remote filename
     req.addProperty(DeliveryPropertyDefinitions.WEBDAV_REMOTE_FIL
ENAME, "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.

| Property | Description |
|---|---|
| WEBDAV_CONTENT_TYPE | Required<br>Enter the document content type (example: "application/pdf"). |
| WEBDAV_HOST | Required<br>Enter the server host name. |
| WEBDAV_PORT | Optional<br>Enter the server port number.<br>Default is 80. |
| WEBDAV_REMOTE_DIRECTORY | Required.<br>Enter the remote directory name (example: "/myreports/"). |

| Property | Description |
|---|---|
| WEBDAV_REMOTE_FILENAME | Required.<br>Enter the remote file name. |
| WEBDAV_AUTHTYPE | Optional<br>Valid values for authentication type are:<br>WEBDAV_AUTHTYPE_NONE - no authentication (default)<br>WEBDAV_AUTHTYPE_BASIC - use HTTP basic authentication<br>WEBDAV_AUTHTYPE_DIGEST - use HTTP digest authentication |
| WEBDAV_USERNAME | Optional<br>Enter the username for HTTP authentication. |
| WEBDAV_PASSWORD | Optional<br>Enter the password for HTTP authentication. |
| WEBDAV_ENCTYPE | Optional<br>Valid values for encryption type are:<br>WEBDAV_ENCTYPE_NONE - no encryption (default)<br>WEBDAV_ENCTYPE_SSL - use Secure Socket Layer |
| WEBDAV_USE_FULL_URL | Optional<br>Set to "true" to send the full URL for the HTTP request header. Valid values are "true" or "false" (default). |
| WEBDAV_USE_CHUNKED_BODY | Optional<br>Valid values are "true" (default) to use HTTP chunked transfer coding for the message body, or "false". |

## Deliver Your Documents Using FTP

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

**Example**

```
// create delivery manager instance
     DeliveryManager dm = new DeliveryManager();
     // create a delivery request
     DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_F
TP);

     // set hostname of the FTP server
     req.addProperty(DeliveryPropertyDefinitions.FTP_HOST, "myftph
ost");
     // 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, "xd
o");
     req.addProperty(DeliveryPropertyDefinitions.FTP_PASSWORD, "xd
o");
     // set the remote directory that you want to send your docume
nt to
     req.addProperty(DeliveryPropertyDefinitions.FTP_REMOTE_DIRECT
ORY, "pub");
     // set the remote file name
     req.addProperty(DeliveryPropertyDefinitions.FTP_REMOTE_FILENA
ME, "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<br>Enter the server host name. |
| FTP_PORT | Optional<br>Enter the server port number. Default is 21. |
| FTP_USERNAME | Required<br>Enter the login user name to the FTP server. |
| FTP_PASSWORD | Required<br>Enter the login password to the FTP server. |
| FTP_REMOTE_DIRECTORY | Required<br>Enter the directory to which to deliver the document (example: /pub/) |
| FTP_REMOTE_FILENAME | Required<br>Enter the document file name for the remote server. |
| FTP_BINARY_MODE | Optional<br>Valid values are "true" (default) or "false". |

## 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_H
TTP);

     // set request method
     req.addProperty(DeliveryPropertyDefinitions.HTTP_METHOD, Deli
veryPropertyDefinitions.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, "myhos
t");
     // set the HTTP server port number
     req.addProperty(DeliveryPropertyDefinitions.HTTP_PORT, "80");
     // set the target remote directory
     req.addProperty(DeliveryPropertyDefinitions.HTTP_REMOTE_DIREC
TORY, "/servlet/");
     // set the remote filename (servlet class)
     req.addProperty(DeliveryPropertyDefinitions.HTTP_REMOTE_FILEN
AME, "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.

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

| Property | Description |
|---|---|
| HTTP_REMOTE_DIRECTORY | Required<br>Enter the remote directory name (example: "/ home/"). |
| HTTP_REMOTE_FILENAME | Required<br>Enter the file name to save the document as in the remote directory. |
| HTTP_AUTHTYPE | Optional<br>Valid values for authentication type are:<br>HTTP_AUTHTYPE_NONE - no authentication (default)<br>HTTP_AUTHTYPE_BASIC - use basic HTTP authentication<br>HTTP_AUTHTYPE_DIGEST - use digest HTTP authentication |
| HTTP_USERNAME | Optional<br>If the server requires authentication, enter the username. |
| HTTP_PASSWORD | Optional<br>If the server requires authentication, enter the password for the username. |
| HTTP_ENCTYPE | Optional<br>Enter the encryption type:<br>HTTP_ENCTYPE_NONE - no encryption (default)<br>HTTP_ENCTYPE_SSL - use Secure Socket Layer |
| HTTP_USE_FULL_URL | Optional<br>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<br>Valid values are "true" (default) to use HTTP chunked transfer coding for the message body, or "false". |
| HTTP_TIMEOUT | Optional<br>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). |

## 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_I
PP_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 9-27.

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_I
PP_PRINTER);

     // set buffering mode
     req.addProperty(DeliveryPropertyDefinitions.BUFFERING_MODE, "
true");
     req.addProperty(DeliveryPropertyDefinitions.TEMP_DIR, "/tmp")
;
           :
           :
           :
     // submit request
     req.submit();
           :
           :
     // submit request again
     req.submit();
           :
           :
     // close the request
     req.close();
```

## Monitoring Delivery Status

The delivery system allows you to check the latest delivery status of your request by calling the **`getStatus()`** method. You can check the status of the request anytime, but currently you must retain the delivery request object. Status definitions are defined in the DeliveryRequest interface.

Monitoring delivery status is not available for the SMTP and HTTP delivery channels.

**Example**

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

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

     // get request status
     int status = req.getStatus();
     if (status == DeliveryRequest.STATUS_SUCCESSFUL)
     {
        System.out.println("Request has been delivered successfull
y.");
     }
           :
           :
     // 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 9-17 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, "t
rue");

     // create delivery requests
     DeliveryRequest req1 = dm.createRequest(DeliveryManager.TYPE_
IPP_PRINTER);
     DeliveryRequest req2 = dm.createRequest(DeliveryManager.TYPE_
IPP_FAX);
     DeliveryRequest req3 = dm.createRequest(DeliveryManager.TYPE_
SMTP_EMAIL);
           :
           :
```

## Delivering Multiple Requests with a Single Output Stream

To deliver your document to multiple delivery channels with a single output stream, use the MultipleRequestHandler utility. Register all your delivery requests to the utility to get a single output stream that internally distributes the data to the requests.

**Example**

```
 // create delivery manager instance
     DeliveryManager dm = new DeliveryManager();
            :
            :
     // create delivery requests
     DeliveryRequest req1 = dm.createRequest(DeliveryManager.TYPE_
IPP_PRINTER);
     DeliveryRequest req2 = dm.createRequest(DeliveryManager.TYPE_
IPP_FAX);
     DeliveryRequest req3 = dm.createRequest(DeliveryManager.TYPE_
SMTP_EMAIL);
            :
            :
     // create MultipleRequestHandler instance
     MultipleRequestHandler mh = new MultipleRequestHandler();
     // register delivery requests
     mh.addRequest(req1);
     mh.addRequest(req2);
     mh.addRequest(req3);
     // get the ouptput stream
     OutputStream out = mh.getDocumentOutputStream();
            :
            :
     // write the document
     out.write(yourDocument);
            :
            :
     // submit all delivery requests
     mh.submitRequests();
            :
            :
     // close requests
     req1.close();
     req2.close();
     req3.close();
```

## Adding a Custom Delivery Channel

You can add custom delivery channels to the system by following the steps below:

1. Define the delivery properties
2. Implement the DeliveryRequest interface
3. Implement the DeliveryRequestHandler interface
4. Implement the DeliveryRequestFactory interface
5. Register your custom DeliveryRequestFactory to the DeliveryManager

The following sections detail how to create a custom delivery channel by creating a sample called "File delivery channel" that delivers documents to the local file system.

### Define Delivery Properties

The first step to adding a custom delivery channel is to define the properties. These will vary depending on what you want your channel to do. You can define constants for your properties. Our example, a file delivery channel requires only one property, which is the destination.

Sample code is:

**Example**

```
package oracle.apps.xdo.delivery.file;

public interface FilePropertyDefinitions
  {
     /** Destination property definition.  */
     public static final String FILE_DESTINATION = "FILE_DESTINATI
ON: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.delvery.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_DESTINATIO
N};

  /**
   * 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 DeliveryRequest
Handler
{

  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_DES
TINATION);
      mOut = new BufferedOutputStream(new FileOutputStream(filenam
e));

      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_SUCCESSF
UL);
      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_DESTI
NATION);
    File f = new File(filename);

    // set request status
    if (f.exists())
      mRequest.setStatus(DeliveryStatusDefinitions.STATUS_SUCCESSF
UL);
    else
      mRequest.setStatus(DeliveryStatusDefinitions.STATUS_FAILED_I
O_ERROR);

  }
  /**
   * returns the request status.
   */
  public boolean isRequestOpen()
  {
    return mIsOpen;
  }

  /**
   * closes the request, frees all resources.
   */
  public void closeRequest()
  {
    mIsOpen = false;
    try
    {
      if (mOut != null)
      {
        mOut.flush();
        mOut.close();
      }
    }
    catch (IOException e)
    {
    }
    finally
    {
      mOut = null;
    }
  }

}
```

### Implement DeliveryRequestFactory Interface

Implement the DeliveryRequestFactory interface to register your custom delivery channel to the delivery system.

A sample implementation of oracle.apps.xdo.delivery.DeliveryRequestFactory is as follows:

**Example**

```
package oracle.apps.xdo.delivery.file;

import oracle.apps.xdo.delivery.DeliveryRequest;
import oracle.apps.xdo.delivery.DeliveryRequestFactory;
import oracle.apps.xdo.delivery.DeliveryRequestHandler;

public class FileDeliveryRequestFactory
implements DeliveryRequestFactory
{
  /**
   * default constructor.
   */
  public FileDeliveryRequestFactory()
  {
  }
  /**
   * returns delivery request.
   */
  public DeliveryRequest createRequest()
  {
    return new FileDeliveryRequest();
  }
  /**
   * returns delivery request handler.
   */
  public DeliveryRequestHandler createRequestHandler()
  {
    return new FileDeliveryRequestHandler();
  }
  /**
   * returns this
   */
  public DeliveryRequestFactory getFactory()
  {
    return this;
  }
}
```

## Register your custom DeliveryRequestFactory to DeliveryManager

The final step is to register your custom delivery channel to the delivery system. You can register your delivery channel in two ways:

- Static method

  Use this method to register your delivery channel to the whole delivery system by specifying it in the configuration file. See Configuration File Support, page 9-27 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.del
ivery.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</u
ri>

      </server>
      <server name="myprinter2" type="ipp_printer" >
        <host>myprinter2.oracle.com</host>
        <port>631</port>

        <uri>ipp://myprinter2.oracle.com:631/printers/myprinter2</u
ri>
        <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.FileDelive
ryRequestFactory</channel>
   </channels>

 </config>
```

## Defining Multiple Servers for a Delivery Channel

You can define multiple server entries for each delivery channel. For example, the preceding sample configuration file has two server entries for the "ipp_printer" delivery channel ("myprinter1" and "myprinter2").

Load a server entry for a delivery request by calling DeliveryRequest.setServer() method. Following is an example:

**Example**

```
// create delivery manager instance
     DeliveryManager dm = new DeliveryManager();
     // create a delivery request
     DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_I
PP_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.

# A
# XML Publisher Configuration File

## XML Publisher Configuration File

You can customize the behavior of XML Publisher by setting properties in a configuration file. The configuration file is optional. There is no default configuration file in the system.

The configuration file is primarily used for:

- Setting a temporary directory
- Setting general properties for PDF files generated by XML Publisher
- Setting security properties for PDF files generated by XML Publisher
- Setting font locations and substitutions

> **Important:** It is **strongly** recommended that you set up this configuration file to create 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 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.

### File Name and Location

You must name this file **`xdo.cfg`** and place it under **`<JRE_TOP>/lib`**.

### 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/confi
g/">

   <!-- Properties -->
   <properties>
      <!-- System level properties -->
      <property name="system-temp-dir">/tmp</property>

      <!--  PDF compression -->
      <property name="pdf-compression">true</property>

      <!--  PDF Security -->
      <property name="pdf-security">true</property>
      <property name="pdf-open-password">user</property>
      <property name="pdf-permissions-password">owner</property>
      <property name="pdf-no-printing">true</property>
      <property name="pdf-no-changing-the-document">true</property
>
   </properties>

   <!-- Font setting -->
   <fonts>
      <!-- Font setting (for FO to PDF etc...) -->
      <font family="Arial" style="normal" weight="normal">
         <truetype path="/fonts/Arial.ttf" />
      </font>
      <font family="Default" style="normal" weight="normal">
         <truetype path="/fonts/ALBANWTJ.ttf" />
      </font>

      <!--Font substitute setting (for PDFForm filling etc...) -->
      <font-substitute name="MSGothic">
         <truetype path="/fonts/msgothic.ttc" ttcno="0" />
      </font-substitute>
   </fonts>
</config>
```

### How to Read the Element Specifications

The following is an example of an element specification:

```
<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.

## Root Element

The `<config>` element is the root element. It has the following structure:

```
<config version="cdata" xmlns="http://xmlns.oracle.com/oxp/config
/">
   <fonts> ... </fonts> [0..n]
   <properties> ... </properties> [0..n]
</config>
```

#### Attributes

| | |
|---|---|
| **version** | The version number of the configuration file format. Specify 1.0.0. |
| **xmlns** | The namespace for XML Publisher's configuration file. Must be **`http://xmlns.oracle.com/oxp/config/`** |

#### Description

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

- Properties (**`<properties>`** elements)
- Font definitions (**`<fonts>`** elements)

The **`<fonts>`** and **`<properties>`** elements can appear multiple times. If conflicting definitions are set up, the last occurrence prevails.

## Properties

This section describes the **`<properties>`** element and the **`<property>`** element.

### The <properties> element

The properties element is structured as follows:

```
<properties locales="cdata">
   <property>...
   </property> [0..n]
</properties>
```

#### Attributes

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

#### Description

The **`<properties>`** element defines a set of properties. You can specify the **`locales`** attribute to define locale-specific properties. Following is an example:

**Example**

```
<!-- Properties for all locales -->
<properties>
 ...Property definitions here...
</properties>

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

### The <property> element

The **`<property`**> element has the following structure:

```
<property name="cdata">
  ...pcdata...
</property>
```

#### Attributes

| | |
|---|---|
| **name** | Specify the property name. |

#### Description

Property is a name-value pair. Specify the property name (key) to the name attribute and the value to the element value. For the available names and values for the **`<property>`** element, see XML Publisher Properties, page A- 9 .

**Example**

```
<properties>
  <property name="system-temp-dir">d:\tmp</property>
  <property name="system-cache-page-size">50</property>
  <property name="pdf-compression">true</property>
</properties>
```

## Font Definitions

Font definitions include the following elements:

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

### <fonts> element

The **`<fonts>`** element is structured as follows:

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

#### Attributes

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

#### Description

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

**Example**

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

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

## <font> element

Following is the structure of the **`<font>`** element:

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

### Attributes

| | |
|---|---|
| **family** | Specify any family name for this font. If you specify "Default" to this attribute, you can define a default fallback font. **family** is case-insensitive. |
| **style** | Specify "normal" or "italic" for the font style. |
| **weight** | Specify "normal" or "bold" for the font weight. |

### Description

Defines an XML Publisher font. This element it primarily used to define font for FO-to-PDF processing. The PDF Form Processor 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

| | |
|---|---|
| **name** | 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>
```

### <truetype> element

The form of the truetype element is as follows:

```
<truetype path="cdata" ttcno="cdata"/>
```

#### Attributes

| | |
|---|---|
| **path** | Specify the absolute path for the font. |
| **ttcno** | Specify the TTC number starting from 0 if the specified font is a TrueType collection (.TTC). This attribute is optional. |

#### Description

<truetype> element defines a TrueType font.

**Example**
```
<!--Define "Arial" font -->
<font family="Arial" style="normal" weight="normal">
  <truetype path="/fonts/Arial.ttf"/>
</font>
```

### <type1> element>

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

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

#### Attributes

| | |
|---|---|
| **name** | 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>
```

## 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

## Predefined Fonts

XML Publisher has several predefined fonts. These fonts do not require any font setting in the configuration file.

To use the predefined TrueType fonts, the TrueType font files must exist under **`<java.home>/lib/fonts`** directory. An example of a **`<java.home>`** get is as follows:

**`StringjavaHome=(String)System.getProperty("java.home");`**
Font family names are case insensitive.

The Type1 fonts are listed in the following table:

| Number | Font Family | Style | Weight | Actual Font |
|---|---|---|---|---|
| 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 |

| Number | Font Family | Style | Weight | Actual Font |
|---|---|---|---|---|
| 4 | Courier | italic | bold | Courier-BoldOblique |
| 5 | Helvetica | normal | normal | Helvetica |
| 5 | Helvetica | normal | bold | Helvetica-Bold |
| 5 | Helvetica | italic | normal | Helvetica-Oblique |
| 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) |

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

## XML Publisher Properties

This section lists the properties that you can set in the XML Publisher Configuration file.

## FO Engine Properties

The following table defines the properties that you can set to control FO Engine processing.

| Property Name | Default Value | Description |
|---|---|---|
| `system-temp-dir` | N/A | Temporary directory for the FO Processor.<br>Setting a temporary directory is **strongly** recommended to avoid "Out of Memory" errors when processing large files. |
| `system-cache-page-size` | 50 | This property is enabled only when you have specified a **`system-temp-dir`**. During table of contents creation, 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 **`system-temp-dir`**. |
| `pdf-compression` | true | Specify "true" or "false" to control compression of the output PDF file. Set to "true" to compress the PDF. |
| `pdf-hide-toolbar` | false | Specify "true" to hide the viewer application's toolbar when the document is active. |
| `pdf-hide-menubar` | false | Specify "true" to hide the viewer application's menu bar when the document is active. |
| `pdf-security` | false | Specify "true" or "false" to control the output PDF security. If you specify "true" the output PDF file will be encrypted.<br>You must also specify the following properties (described below):<br>• **`pdf-open-password`**<br>• **`pdf-permissions-password`**<br>• **`pdf-encryption-level`**<br>**Example**:<br>**`<property name="pdf-security">true</property>`**<br>**`<property name="pdf-open-password">user</property>`**<br>**`<property name="pdf-permissions-password">owner</property>`**<br>**`<property name="pdf-encryption-level">0</property>`** |
| `pdf-open-password` | (null) | This property is effective only when the **`pdf-security`** property is set to "true".<br>Specify the password to open the output PDF. This password enables users to open the document only.<br>Setting this property in the configuration file is not recommended because the configuration file is not encrypted. |
| `pdf-permissions-password` | (null) | This property is effective only when the **`pdf-security`** property is set to "true".<br>Specify the permissions password for the output PDF file. This password enables a user to override the security setting.<br>Setting this property in the configuration file is not recommended because the configuration file is not encrypted. |

| Property Name | Default Value | Description |
|---|---|---|
| **`pdf-permissions`** | 0 | This property is effective only when the **`pdf-security`** property is set to "true".<br>Specify the security permissions for the output PDF. For more information see section 3.5 of Adobe's *PDF Reference: Version 1.4* (third edition).<br>If this property is not set, the following permission properties are used instead (see below for descriptions of each property): |
| **`pdf-encryption-level`** | 0 | Specify the encryption level for the output PDF file. The possible values are:<br>• 0: Low (40-bit RC4, Acrobat 3.0 or later)<br>• 1: High (128-bit RC4, Acrobat 5.0 or later)<br>This property is effective only when the **`pdf-security`** property is set to "true".<br>When **`pdf-encryption-leve`**l is 0 you can also set the following properties (described below):<br>• **`pdf-no-printing`**<br>• **`pdf-no-changing-the-document`**<br>• **`pdf-no-cceda`**<br>• **`pdf-no-accff`**<br>When **`pdf-encryption-level`** is 1, you can also set the following properties (described below):<br>• **`pdf-enable-accessibility`**<br>• **`pdf-enable-copying`**<br>• **`pdf-changes-allowed`**<br>• **`pdf-printing-allowed`** |
| **`pdf-no-printing`** | false | Permission when **`pdf-encryption-level`** is set to 0 (40-bit). When set to "true" printing is disabled for the PDF file. |
| **`pdf-no-changing-the-document`** | false | Permission when **`pdf-encryption-level`** is set to 0 (40-bit). When set to "true" the PDF file cannot be edited. |
| **`pdf-no-cceda`** | false | Permission when **`pdf-encryption-level`** is set to 0 (40-bit). When set to "true", content copying, content extraction, and accessibility features are disabled. |
| **`pdf-enable-accessibility`** | true | Permission when **`pdf-encryption-level`** is set to 1 (128-bit). When set to "true", text access for screen reader devices is enabled. |
| **`pdf-enable-copying`** | false | Permission when **`pdf-encryption-level`** is set to 1 (128-bit). When set to "true", copying of text, images, and other content is enabled. |

| Property Name | Default Value | Description |
|---|---|---|
| **pdf-changes-allowed** | 0 | Permission when **pdf-encryption-level** is set to 1 (128-bit). Valid values are:<br>• 0: none<br>• 1: Allows inserting, deleting, and rotating pages<br>• 2: Allows filling in form fields and signing<br>• 3: Allows commenting, filling in form fields, and signing<br>• 4: Allows all changes except extracting pages |
| **pdf-printing-allowed** | 0 | Permission when **pdf-encryption-level** is set to 1 (128-bit). Valid values are:<br>• 0: None<br>• 1: Low Resolution (150 dpi)<br>• High Resolution |
| **font.<family>.<style>.<weight>** | N/A | Defines a font for XML Publisher.<br>**<family>** is case insensitive.<br>**<style>** must be "italic" or "normal"<br>**<weight>** must be "bold" or "normal"<br>The value of this property can be one of the following:<br>• **type1.<fontname>**<br>• **truetype.<path>d**<br>• **truetype.<path>.(<ttcno>)**<br>**Example**<br>`<font family="Courier" style="italic" weight="bold">`<br>`<type1 name="Courier-BoldOblique"/>`<br>`</font>`<br>is same as:<br>`font.Courier.italic.bold=type1.Courier-BoldOblique`<br>**Example**<br>`<font family="arial" style="normal" weight="normal">`<br>`<truetype path="C:\fonts\Arial.ttf"/>`<br>`</font>`<br>is same as:<br>`font.Arial.normal.normal=truetype.C:\fonts\Arial.ttf`<br>**Example**<br>`<font family="MSGothic" style="normal" weight="normal">`<br>`<truetype path="C:\fonts\msgothic.ttc" ttcno="0"/>`<br>`</font>`<br>is same as:<br>`font.MSGothic.normal.normal="truetype.C:\fonts\msgothic.ttc(0)` |

## PDF Form Processor Properties

**Property Name:**

**`font-substitute.<fontname>`**

**Description:**

Use this property to map a font name from the PDF to a font file for I18N support.

The value of this property is one of the following:

- **`truetype.<path>`**
- **`truetype.<path>.<ttcno>`**

Example 1:

```
<font-substitute name="Courier">
   <truetype path="C:\fonts\Arial.ttf />
</font-substitute>
```

is same as:

```
font-substitute.Courier=truetype.C:\fonts\Arial.ttf
```

Example 2:

```
<font-substitute name="Courier">
   <truetype patch="C:\fonts\msgothic.ttc" ttcno="0"/>
</font-substitute>
```

is same as:

```
font-substitute.Courier=truetype.C:\fonts\msgothic.ttc(0)
```

# Index