Oracle® XML Publisher
Report Designer's Guide
Release 12
Part No. B31410-01

December 2006
The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

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

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

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

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

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

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

Send Us Your Comments

Preface

1 XML Publisher Introduction
   Introduction .............................................................................................................................. 1-1
   Process Overview ............................................................................................................... 1-3
   Structure of the XML Publisher Documentation Set ...................................................... 1-4

2 Creating an RTF Template
   Introduction ............................................................................................................................... 2-2
   Supported Modes ................................................................................................................ 2-2
   Prerequisites ........................................................................................................................ 2-2
   Overview ................................................................................................................................... 2-3
      Using the Business Intelligence Publisher Template Builder for Microsoft Word Add-in .... 2-3
      Associating the XML Data to the Template Layout ....................................................... 2-4
   Designing the Template Layout .......................................................................................... 2-7
   Adding Markup to the Template Layout .............................................................................. 2-8
      Creating Placeholders ..................................................................................................... 2-8
      Defining Groups ................................................................................................................ 2-12
   Defining Headers and Footers ......................................................................................... 2-16
      Native Support .................................................................................................................. 2-16
   Images and Charts .............................................................................................................. 2-18
      Images ................................................................................................................................ 2-18
      Chart Support ................................................................................................................. 2-20
   Drawing, Shape and Clip Art Support ................................................................................ 2-31
Supported Native Formatting Features .................................................... 2-42
  General Features .................................................................................. 2-42
  Alignment .............................................................................................. 2-43
  Tables ..................................................................................................... 2-43
  Date Fields ............................................................................................. 2-46
  Multicolumn Page Support ................................................................... 2-46
  Background and Watermark Support .................................................... 2-48

Template Features ................................................................................ 2-50
  Page Breaks .......................................................................................... 2-50
  Initial Page Number .............................................................................. 2-51
  Last Page Only Content ....................................................................... 2-52
  End on Even or End on Odd Page ......................................................... 2-55
  Hyperlinks ............................................................................................. 2-55
  Table of Contents ................................................................................ 2-58
  Generating Bookmarks in PDF Output ................................................ 2-58
  Check Boxes ........................................................................................ 2-59
  Drop Down Lists ................................................................................ 2-60

Conditional Formatting ........................................................................ 2-63
  If Statements ......................................................................................... 2-64
    If Statements in Boilerplate Text .......................................................... 2-64
  If-then-Else Statements ....................................................................... 2-65
  Choose Statements .............................................................................. 2-66
  Column Formatting ............................................................................... 2-67
  Row Formatting .................................................................................... 2-70
  Cell Highlighting .................................................................................. 2-72

Page-Level Calculations ........................................................................ 2-74
  Displaying Page Totals ......................................................................... 2-74
  Brought Forward/Carried Forward Totals ............................................ 2-76
  Running Totals ...................................................................................... 2-80

Data Handling ....................................................................................... 2-82
  Sorting .................................................................................................. 2-82
  Checking for Nulls ............................................................................... 2-82
  Regrouping the XML Data .................................................................... 2-83

Variables, Parameters, and Properties .................................................. 2-89
  Using Variables .................................................................................... 2-89
  Defining Parameters ............................................................................ 2-90
  Setting Properties ............................................................................... 2-92

Advanced Report Layouts ..................................................................... 2-95
  Batch Reports ....................................................................................... 2-95
  Cross-Tab Support ............................................................................... 2-96
  Dynamic Data Columns ....................................................................... 2-99
Send Us Your Comments

Oracle XML Publisher Report Designer's Guide, Release 12
Part No. B31410-01

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

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

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

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

Send your comments to us using the electronic mail address: appsdoc_us@oracle.com

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

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

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

Intended Audience

Welcome to Release 12 of the Oracle XML Publisher Report Designer’s Guide.

This guide is intended for users responsible for designing report layout templates for use with XML Publisher. It assumes you have a working knowledge of the following:

- The principles and customary practices of your business area.
- Computer desktop application usage and terminology.

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

See Related Information Sources on page x for more Oracle Applications product information.

TTY Access to Oracle Support Services

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

Documentation Accessibility

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

Accessibility of Code Examples in Documentation

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

Accessibility of Links to External Web Sites in Documentation

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

Structure

1 XML Publisher Introduction
2 Creating an RTF Template
3 Creating a PDF Template
4 Creating an eText Template
5 XML Publisher Extended Functions
A Supported XSL-FO Elements

Related Information Sources

This book is included on the Oracle Applications Documentation Library, which is supplied in the Release 12 Media Pack. You can download soft-copy documentation as PDF files from the Oracle Technology Network at http://otn.oracle.com/documentation, or you can purchase hard-copy documentation from the Oracle Store at http://oraclestore.oracle.com. The Oracle Applications Release 12 Documentation Library contains the latest information, including any documents that have changed significantly between releases. If substantial changes to this book are necessary, a revised version will be made available on the "virtual" documentation library on Oracle MetaLink.

For a full list of documentation resources for Oracle Applications Release 12, see Oracle Applications Documentation Resources, Release 12, OracleMetaLink Document 394692.1.

If this guide refers you to other Oracle Applications documentation, use only the Release 12 versions of those guides.

Online Documentation

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

- **Online Help** - Online help patches (HTML) are available on OracleMetaLink.
• **PDF Documentation** - See the Oracle Applications Documentation Library for current PDF documentation for your product with each release. The Oracle Applications Documentation Library is also available on OracleMetaLink and is updated frequently.

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

**Oracle Applications User's Guide**

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

**Oracle Applications Developer's Guide**

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

**Oracle Applications Flexfields Guide**

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

**Oracle Application Framework Developer's Guide**

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

**Oracle Applications Concepts**

This book is intended for all those planning to deploy Oracle E-Business Suite Release 12, or contemplating significant changes to a configuration. After describing the Oracle Applications architecture and technology stack, it focuses on strategic topics, giving a
broad outline of the actions needed to achieve a particular goal, plus the installation and configuration choices that may be available.

**Oracle Applications System Administrator’s Guide Documentation Set**

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

**Oracle Applications Multiple Organizations Implementation Guide**

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

**Integration Repository**

The Oracle Integration Repository is a compilation of information about the service endpoints exposed by the Oracle E-Business Suite of applications. It provides a complete catalog of Oracle E-Business Suite’s business service interfaces. The tool lets users easily discover and deploy the appropriate business service interface for integration with any system, application, or business partner.

The Oracle Integration Repository is shipped as part of the E-Business Suite. As your instance is patched, the repository is automatically updated with content appropriate for the precise revisions of interfaces in your environment.

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

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

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

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

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

- Introduction
- Process Overview
- Structure of the XML Publisher Documentation Set

Introduction

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

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

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

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

User Interfaces

XML Publisher provides the Template Manager for E-Business Suite users to register and maintain report templates and link them to data sources. Once both have been registered, simply run the report using standard request submission and XML Publisher will merge the template and its data source into the customized report.

Template Manager

The Template Manager is the repository for your templates and data sources. It is also the interface for to 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 and translations. For a complete description of the Template Manager, see the Oracle XML Publisher Administration and Developer’s Guide.
XML Publisher Desktop Components

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

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

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

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.

2. Design your template.
   Your template files can be either in Rich Text Format (RTF) or Portable Document Format (PDF).
   RTF is a specification used by many word processing applications, such as Microsoft Word. You design the template using your desktop word processing application and save the file as an RTF file type (.rtf extension). Insert basic markup tags to the document to prepare it for merging with the XML data. XML Publisher recognizes the formatting features that you apply and converts them to XSL-FO.
   Use Adobe Acrobat to apply markup tags to your custom-designed or downloaded PDF template.

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

4. Add desired translations of your RTF template.
   XML Publisher's translation utility allows you to extract the translatable strings from your template into an XLIFF file. Translate the strings in this file and reupload to the Template Manager to make the translation available at runtime.
Runtime
1. Set the concurrent program to generate XML.
2. Run the concurrent program using standard request submission.
   • 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.

Structure of the XML Publisher Documentation Set
The XML Publisher documentation set contains the following two guides: Oracle XML Publisher Report Designer’s Guide and the Oracle XML Publisher Administration and Developer’s Guide.

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

Oracle XML Publisher Report Designer’s Guide
Creating an RTF Template - describes how to use your word processing application in conjunction with your report XML file to create a customized template for the report.
Creating a PDF Template - describes how to use Adobe Acrobat in conjunction with your report XML file to create a customized template in PDF.
Creating an eText Template - describes how to create a table-based template to comply with EDI and EFT file specifications. These templates are processed by the eText Processing Engine to create flat text files for exchange with business partners.
XML Publisher Extended Functions - lists SQL and XSL functions that XML Publisher has extended.
Supported XSL-FO Elements - lists the FO elements supported by the XML Publisher engines.

Oracle XML Publisher Administration and Developer’s Guide
This guide includes information on setting up XML Publisher, running reports, using the data engine, and leveraging the APIs.
Using the Template Manager - (pertains to E-Business Suite customers only) describes how to register your Oracle report as a data definition and upload your templates to the Template Manager.
Generating Your Customized Output - (pertains to E-Business Suite customers only) describes how to submit your report request using the concurrent manager to generate output in your customized template.
Administration - describes the Administration interface that allows you to set configuration properties, upload fonts, create font mappings, and create currency mappings.

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

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

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

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

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

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

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

This chapter covers the following topics:

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

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

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

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

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

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

Supported Modes

XML Publisher supports two methods for creating RTF templates:

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

- Form Field Method

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

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

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

Prerequisites

Before you design your template, you must:
Creating an RTF Template

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

Overview

Creating an RTF template file for use with XML Publisher consists of the following steps:

1. Generate sample data from your report.
   You must have sample data either to reference while designing the report manually, or to load to the XML Publisher Template Builder for Word Add-in.

2. Load the data to the Template Builder for Word Add-in and use its features to add data fields, tables, charts, and other report items to your template.
   Alternatively, insert the XML Publisher tags manually into your template, using the guidelines in this chapter.

3. Upload the template to the Template Manager to make it available to XML Publisher at runtime.

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

Using the Business Intelligence Publisher Template Builder for Microsoft Word Add-in

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

**Note:** The BI Publisher Template Builder for Word Add-in includes features to log in to and interact with Oracle Business Intelligence Publisher Enterprise. These features only work with the Oracle BI Publisher Enterprise or Oracle BI Enterprise Edition implementations. See the Template Builder help for more information.

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

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

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

**Associating the XML Data to the Template Layout**

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

**Sample Template Layout**

![Sample Template Layout](image)

Supplier:

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

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
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
<?xml version="1.0" encoding="WINDOWS-1252" ?>
<VENDOR_REPORT>
  <LIST_G_VENDOR_NAME>
    <G_VENDOR_NAME>
      <VENDOR_NAME>COMPANY A</VENDOR_NAME>
    </G_VENDOR_NAME>
  </LIST_G_VENDOR_NAME>
  <LIST_G_INVOICE_NUM>
    <G_INVOICE_NUM>
      <SET_OF_BOOKS_ID>124</SET_OF_BOOKS_ID>
      <GL_DATE>10-NOV-03</GL_DATE>
      <INV_TYPE>Standard</INV_TYPE>
      <INVOICE_NUM>031110</INVOICE_NUM>
      <INVOICE_DATE>10-NOV-03</INVOICE_DATE>
      <INVOICE_CURRENCY_CODE>EUR</INVOICE_CURRENCY_CODE>
      <ENT_AMT>122</ENT_AMT>
      <ACCTD_AMT>122</ACCTD_AMT>
      <VAT_CODE>VAT22%</VAT_CODE>
    </G_INVOICE_NUM>
  </LIST_G_INVOICE_NUM>
  <ENT_SUM_VENDOR>1000.00</ENT_SUM_VENDOR>
  <ACCTD_SUM_VENDOR>1000.00</ACCTD_SUM_VENDOR>
</G_VENDOR_NAME>
</LIST_G_VENDOR_NAME>
</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 2-83.

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 2-16.

For a detailed list of supported formatting features in Microsoft Word, see Supported
Native Formatting Features, page 2-42. Additional formatting and reporting features are described at the end of this section.

**Adding Markup to the Template Layout**

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

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

XML Publisher provides tags to add markup to your template.

*Note:* For the XSL equivalents of the XML Publisher tags, see XSL Equivalent Syntax, page 5-6.

**Creating Placeholders**

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

Enter placeholders in your document using the following syntax:

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

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

There are two ways to insert placeholders in your document:

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

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

**Basic RTF Method**

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

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

```xml
<?XML element tag name?>
```
In the example, the template field "Supplier" maps to the XML element VENDOR_NAME. In your document, enter:

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

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

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

<table>
<thead>
<tr>
<th>Invoice Num</th>
<th>Invoice</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Total for Supplier:</td>
<td></td>
</tr>
</tbody>
</table>

### Form Field Method

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

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

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

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

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

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

The following figure shows the **Text Form Field Options** dialog box and the **Form Field Help Text** dialog box with the appropriate entries for the Supplier field.
**Tip:** For longer strings of XML Publisher syntax, use the Help Key (F1) tab instead of the Status Bar tab. The text entry field on the Help Key (F1) tab allows more characters.

8. Select **OK** to apply.

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

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

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

<table>
<thead>
<tr>
<th>Template Field Name</th>
<th>Default Text Entry (Form Field Method)</th>
<th>Placeholder Entry (XML Tag Name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invoice Num</td>
<td>1234566</td>
<td>&lt;?INVOICE_NUM?&gt;</td>
</tr>
<tr>
<td>Invoice Date</td>
<td>1-Jan-2004</td>
<td>&lt;?INVOICE_DATE?&gt;</td>
</tr>
<tr>
<td>GL Date</td>
<td>1-Jan-2004</td>
<td>&lt;?GL_DATE?&gt;</td>
</tr>
<tr>
<td>Curr</td>
<td>USD</td>
<td>&lt;?INVOICE_CURRENCY_CODE?&gt;</td>
</tr>
<tr>
<td>Entered Amt</td>
<td>1000.00</td>
<td>&lt;?ENT_AMT?&gt;</td>
</tr>
<tr>
<td>Accounted Amt</td>
<td>1000.00</td>
<td>&lt;?ACCTD_AMT?&gt;</td>
</tr>
<tr>
<td>(Total of Entered Amt column)</td>
<td>1000.00</td>
<td>&lt;?ENT_SUM_VENDOR?&gt;</td>
</tr>
<tr>
<td>(Total of Accounted Amt column)</td>
<td>1000.00</td>
<td>&lt;?ACCTD_SUM_VENDOR?&gt;</td>
</tr>
</tbody>
</table>
The following figure shows the Payables Invoice Register with the completed form field placeholder markup.

See the Payables Invoice Register with Completed Basic RTF Markup, page 2-13 for the completed basic RTF markup.

---

**Defining Groups**

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

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

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

Insert the following tag before the first element:

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

Insert the following tag after the final element:

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

**Grouping scenarios**

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

- If you insert the grouping tags around text or formatting elements, the text and formatting elements between the group tags will be repeated.

- If you insert the tags around a table, the table will be repeated.

- If you insert the tags around text in a table cell, the text in the table cell between the tags will be repeated.

- If you insert the tags around two different table cells, but in the same table row, the single row will be repeated.

- If you insert the tags around two different table rows, the rows between the tags will be repeated (this does not include the row that contains the "end group" tag).

**Basic RTF Method**

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

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

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

before the Supplier field that you previously created.

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

The following figure shows the Payables Invoice Register with the basic RTF grouping and placeholder markup:
1. Insert a form field to designate the beginning of the group.

In the help text field enter:

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

Suppliers <?VENDOR_NAME?>
```

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

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

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

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

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

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

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

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

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

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

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

Native Support

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

Inserting Placeholders in the Header and Footer

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

Multiple or Complex Headers and Footers

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

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

<?start:body?>
<?end body?>
Use the tags either directly in the template, or in form fields.

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

1. Insert `<start:body>` before the Suppliers group tag: `<for-each:G_VENDOR_NAME>`

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

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

![Payables Invoice Register](image)

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

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

1. Select Page Setup from the File menu.

2. In the Page Setup dialog, select the Layout tab.

3. In the Headers and footers region of the dialog, select the appropriate check box:
   - Different odd and even
   - Different first page

4. Insert your headers and footers into your template as desired.

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

Images

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

Direct Insertion

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

URL Reference

1. Insert a dummy image in your template.
2. In Microsoft Word's Format Picture dialog box select the Web tab. Enter the following syntax in the Alternative text region to reference the image URL:
   url: '{http://image location}'
   For example, enter:
   url: '{http://www.oracle.com/images/ora_log.gif}'

OA Media Directory Reference

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

1. Insert a dummy image in your template.
2. In Microsoft Word's Format Picture dialog box select the Web tab. Enter the following syntax in the Alternative text region to reference the OA_MEDIA directory:
   url: '${OA_MEDIA}/image name'
   For example, enter:
   url: '${OA_MEDIA}/ORACLE_LOGO.gif'

Element Reference from XML File

1. Insert a dummy image in your template.
2. In Microsoft Word's Format Picture dialog box select the Web tab. Enter the following syntax in the Alternative text region to reference the image URL:
   url: '{IMAGE_LOCATION}'}
where IMAGE_LOCATION is an element from your XML file that holds the full URL to the image.

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

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

where SERVER, IMAGE_DIR, and IMAGE_FILE are element names from your XML file that hold the values to construct the URL.

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

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

Rendering an Image Retrieved from BLOB Data

If your data source is a Data Template (for information, see Data Templates, Oracle XML Publisher Administration and Developer's Guide) and your results XML contains image data that had been stored as a BLOB in the database, use the following syntax in a form field inserted in your template where you want the image to render at runtime:

```
<fo:instream-foreign-object content type="image/jpg">
<xsl:value-of select="IMAGE_ELEMENT"/>
</fo:instream-foreign-object>
```

where

- image/jpg is the MIME type of the image (other options might be: image/gif and image/png)

and

- IMAGE_ELEMENT is the element name of the BLOB in your XML data.

Note that you can specify height and width attributes for the image to set its size in the published report. XML Publisher will scale the image to fit the box size that you define. For example, to set the size of the example above to three inches by four inches, enter the following:

```
<fo:instream-foreign-object content type="image/jpg" height="3 in" width="4 in">
<xsl:value-of select="IMAGE_ELEMENT"/>
</fo:instream-foreign-object>
```

Specify in pixels as follows:

```
<fo:instream-foreign-object content type="image/jpg" height="300 px" width="4 px">
... 
</fo:instream-foreign-object>
```

or in centimeters:

```
<fo:instream-foreign-object content type="image/jpg" height="3 cm" width="4 cm">
... 
</fo:instream-foreign-object>
```

or as a percentage of the original dimensions:
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 following summarizes the steps to add a chart to your template. These steps will be discussed in detail in the example that follows:

1. Insert a dummy image in your template to define the size and position of your chart.

2. Add the definition for the chart to the Alternative text box of the dummy image. The chart definition requires XSL commands.

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

Adding a Sample Chart

Following is a piece of XML data showing total sales by company division.
<sales year=2004>
  <division>
    <name>Groceries</name>
    <totalsales>3810</totalsales>
    <costofsales>2100</costofsales>
  </division>
  <division>
    <name>Toys</name>
    <totalsales>2432</totalsales>
    <costofsales>1200</costofsales>
  </division>
  <division>
    <name>Cars</name>
    <totalsales>6753</totalsales>
    <costofsales>4100</costofsales>
  </division>
  <division>
    <name>Hardware</name>
    <totalsales>2543</totalsales>
    <costofsales>1400</costofsales>
  </division>
  <division>
    <name>Electronics</name>
    <totalsales>5965</totalsales>
    <costofsales>3560</costofsales>
  </division>
</sales>

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

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

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

**Inserting the Dummy Image**

The first step is to add a dummy image to the template in the position you want the chart to appear. The image size will define how big the chart image will be in the final document.
**Important:** You must insert the dummy image as a "Picture" and not any other kind of object.

The following figure shows an example of a dummy image:

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

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

**Adding Code to the Alternative Text Box**
The following graphic shows an example of the XML Publisher code in the **Format Picture Alternative text** box:
The content of the Alternative text represents the chart that will be rendered in the final document. For this chart, the text is as follows:

chart:
  <Graph graphType = "BAR_VERT_CLUST">
  <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>
      </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 <Graph> element. This element has an attribute to define the chart type: graphType. If this attribute is not declared, the default chart is a vertical bar chart. BI Beans supports many different chart types. Several more types are presented in this section. For a complete listing, see the BI Beans graph DTD documentation.

The following code section defines the chart type and attributes:

```xml
<Title text="Company Sales 2004" visible="true" horizontalAlignment="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:

```xml
<Title text="{CHARTTITLE}" visible="true" horizontalAlignment="CENTER"/>
```

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

The next section defines the column and row labels:

```xml
<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>
</LocalGridData>
```

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

```
colCount="{count(//division)}"
```

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

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

The column labels for this example are derived from the data: Groceries, Toys, Cars, and so on. This is done using a for-each loop:
This code loops through the <division> group and inserts the value of the <name> element into the <Label> tag. At runtime, this will generate the following XML:

```
<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>
    <Cell>3810</Cell>
    <Cell>2432</Cell>
    <Cell>6753</Cell>
    <Cell>2543</Cell>
    <Cell>5965</Cell>
  </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 shown in the following figure:
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:

```xml
<Graph graphType = "BAR_HORIZ_CLUST">
  <Title text="Company Sales 2004" visible="true" horizontalAlignment="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, the `rowCount` attribute for the
LocalGridData element is set to 2. Also note the DataValues section defines two sets of data: one for Total Sales and one for Cost of Sales.

**Changing the Appearance of Your Chart**

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

![Company Sales 2004 Chart](image)

The code to support this is as follows:
The colors for the bars are defined in the `SeriesItems` 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>
```
Drawing, Shape and Clip Art Support

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

The following AutoShape categories are supported:

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

Freehand Drawing

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

Hyperlinks

You can add hyperlinks to your shapes. See Hyperlinks, page 2-55.

Layering

You can layer shapes on top of each other and use the transparency setting in Microsoft Word to allow shapes on lower layers to show through. The following graphic shows an example of layered shapes:
3-D Effects

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

Microsoft Equation

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

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

Organization Chart

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

WordArt

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

Data Driven Shape Support

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

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

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

Placement of Commands

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

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

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

where

for-each@shape opens the for-each loop for the shape context

SHAPE_GROUP is the name of the repeating element from the XML file. For each occurrence of the element SHAPE_GROUP a new shape will be created.

shape-offset-y: is the command to offset the shape along the y-axis.

(position()-1)*100) sets the offset in pixels per occurrence. The XSL position command returns the record counter in the group (that is 1,2,3,4); one is subtracted from that number and the result is multiplied by 100. Therefore for the first occurrence the offset would be 0; (1-1) * 100. The offset for the second occurrence would be 100 pixels: (2-1) *100. And for each subsequent occurrence the offset would be another 100 pixels down the page.
Add Text to a Shape

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

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

where SHAPETEXT is the element name in the XML data. At runtime the text will be inserted into the shape.

Add Text Along a Path

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

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

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

Moving a Shape

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

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

Rotating a Shape

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

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

where

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

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

```xml
<shape-rotate:60,'right/bottom'/>
```

You can also specify an x,y coordinate within the shape itself about which to rotate.

### Skewing a Shape

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

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

where

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

POSITION is the point about which to carry out the rotation, for example, 'left/top'. Valid values are combinations of left, right, or center with center, top, or bottom. See the figure under Rotating a Shape, page 2-35. The default is 'left/top'.

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

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

### Changing the Size of a Shape

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

```xml
<shape-size:RATIO/>
```

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

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

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

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

Combining Commands

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

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

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

This would generate the output shown in the following figure:

![CD Ratings Example](image)

CD Ratings Example

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

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

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

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

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

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

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

**Grouped Shape Example**

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

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

![Composite shape](image)

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

The following commands are entered into the Web tab:

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

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

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

`<shape-text:REGION?>`
At runtime the value of the REGION element will appear in the rectangle.

All of these shapes were then grouped together and in the Web tab for the grouped object, the following syntax is added:

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

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

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

![Shapes rendered across the page](image)

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

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

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

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

General Features

- Large blocks of text

- Page breaks

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

1. Place the cursor just before the Supplier group’s closing <end for-each?> tag.

2. Press Ctrl-Enter to insert a page break.

At runtime each Supplier will start on a new page.

Using this Microsoft Word native feature will cause a single blank page to print at the end of your report output. To avoid this single blank page, use XML Publisher’s page break alias. See Special Features: Page Breaks, page 2-50.
• 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.

• Hidden text

You can format text as "hidden" in Microsoft Word and the hidden text will be maintained in RTF output reports.

Alignment

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

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

Tables

Supported table features include:

• Nested Tables

• Cell Alignment

You can align any object in your template using your word processing application’s alignment tools. This alignment will be reflected in the final report output.

• Row spanning and column spanning

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

1. Select the cells you wish to merge.

2. From the Table menu, select Merge Cells.

3. Align the data within the merged cell as you would normally.

At runtime the cells will appear merged.

• Table Autoformatting

XML Publisher recognizes the table autoformats available in Microsoft Word.
1. Select the table you wish to format.

2. From the Table menu, select Autoformat.

3. Select the desired table format.

At runtime, the table will be formatted using your selection.

- **Cell patterns and colors**
  You can highlight cells or rows of a table with a pattern or color.
  1. Select the cell(s) or table.
  2. From the Table menu, select Table Properties.
  3. From the Table tab, select the Borders and Shading... button.
  4. Add borders and shading as desired.

- **Repeating table headers**
  If your data is displayed in a table, and you expect the table to extend across multiple pages, you can define the header rows that you want to repeat at the start of each page.
  1. Select the row(s) you wish to repeat on each page.
  2. From the Table menu, select Heading Rows Repeat.

- **Prevent rows from breaking across pages.**
  If you want to ensure that data within a row of a table is kept together on a page, you can set this as an option using Microsoft Word's Table Properties.
  1. Select the row(s) that you want to ensure do not break across a page.
  2. From the Table menu, select Table Properties.
  3. From the Row tab, deselect the check box "Allow row to break across pages".

- **Fixed-width columns**
  To set the widths of your table columns:
  1. Select a column and then select Table >Table Properties.
  2. In the Table Properties dialog, select the Column tab.
  3. Enable the Preferred width checkbox and then enter the width as a Percent or
in Inches.

4. Select the Next Column button to set the width of the next column.

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

The following figure shows the Table Properties dialog:

![Table Properties dialog](image)

- Text truncation

By default, if the text within a table cell will not fit within the cell, the text will be wrapped. To truncate the text instead, use the table properties dialog.

1. Place your cursor in the cell in which you want the text truncated.

2. Right-click your mouse and select Table Properties... from the menu, or navigate to Table > Table Properties...

3. From the Table Properties dialog, select the Cell tab, then select Options...

4. Deselect the Wrap Text check box.

The following figure shows the Cell Options dialog.
An example of truncation is shown in the following graphic:

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

**Date Fields**

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

**Multicolumn Page Support**

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

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

**Multicolumn Page Example: Labels**

To generate address labels in a two-column format:

1. Divide your page into two columns using the Columns command.

2. Define the repeatable group in the first column. Note that you define the repeatable group only in the first column, as shown in the following figure:

   ![Multicolumn Page Example: Labels](image)

   **Tip:** To prevent the address block from breaking across pages or columns, embed the label block inside a single-celled table. Then specify in the Table Properties that the row should not break across pages. See Prevent rows from breaking across pages, page 2-44.
This template will produce the following multicolumn output:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Big Co</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>1221</td>
</tr>
<tr>
<td>City</td>
<td>Helsinki</td>
</tr>
<tr>
<td>State</td>
<td>FI</td>
</tr>
<tr>
<td>Zip Code</td>
<td>Il...Il...Il...Il...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>My Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>1220</td>
</tr>
<tr>
<td>City</td>
<td>Espoo</td>
</tr>
<tr>
<td>State</td>
<td>FI</td>
</tr>
<tr>
<td>Zip Code</td>
<td>Il...Il...Il...Il...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Small Co</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>1221</td>
</tr>
<tr>
<td>City</td>
<td>Helsinki</td>
</tr>
<tr>
<td>State</td>
<td>FI</td>
</tr>
<tr>
<td>Zip Code</td>
<td>Il...Il...Il...Il...</td>
</tr>
</tbody>
</table>

**Background and Watermark Support**

XML Publisher supports the "Background" feature in Microsoft Word. You can specify a single, graduated color or an image background for your template to be displayed in the PDF output. Note that this feature is supported for PDF output only.

To add a background to your template, use the Format > Background menu option.

**Add a Background Using Microsoft Word 2000**

From the Background pop up menu, you can:

- Select a single color background from the color palette
- Select Fill Effects to open the Fill Effects dialog. The Fill Effects dialog is shown in the following figure:
From this dialog select one of the following supported options:

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

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

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

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

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

Page Breaks

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

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

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

**Example**

For the following XML, assume you want to create a page break for each new supplier:
Creating an RTF Template

In the template sample shown in the following figure, the field called PageBreak contains the split-by-page-break syntax:

<table>
<thead>
<tr>
<th>Supplier</th>
<th>Supplier 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invoice Number</td>
<td>Invoice Date</td>
</tr>
<tr>
<td>10001-1</td>
<td>1-Jan-2005</td>
</tr>
</tbody>
</table>

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

Initial Page Number

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

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

`<?initial-page-number:pagename?>`

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

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

If your XML data contains an element to carry the initial page number, for example:
Enter the following in your template:
<?initial-page-number:PAG\$START?>

Your initial page number will be the value of the PAG\$START element, which in this case is 200.

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

If you define a parameter called PAG\$START, you can pass the initial value by calling the parameter.

Enter the following in your template:
<?initial-page-number:$PAG\$START?>

*Note:* You must first declare the parameter in your template. See Defining Parameters in Your Template, page 2-90.

---

**Last Page Only Content**

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

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

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

1. Create a section break in your template to ensure the content of the final page is separated from the rest of the report.

2. Insert the following syntax on the final page:
   
   ```xml
   <?start@last-page:body?>
   <?end body?>
   ```

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

**Example**

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

```xml
<?xml version="1.0" encoding="WINDOWS-1252"?>
<INVOICELIST>
  <VENDOR>
    <VENDOR_NAME>Nuts and Bolts Limited</VENDOR_NAME>
    <ADDRESS>1 El Camino Real, Redwood City, CA 94065</ADDRESS>
    <INVOICE>
      <INV_TYPE>Standard</INV_TYPE>
      <INVOICE_NUM>981110</INVOICE_NUM>
      <INVOICE_DATE>10-NOV-04</INVOICE_DATE>
      <INVOICE_CURRENCY_CODE>EUR</INVOICE_CURRENCY_CODE>
      <ENT_AMT>122</ENT_AMT>
      <ACCTD_AMT>122</ACCTD_AMT>
      <VAT_CODE>VAT22%</VAT_CODE>
    </INVOICE>
    <INVOICE>
      <INV_TYPE>Standard</INV_TYPE>
      <INVOICE_NUM>100000</INVOICE_NUM>
      <INVOICE_DATE>28-MAY-04</INVOICE_DATE>
      <INVOICE_CURRENCY_CODE>FIM</INVOICE_CURRENCY_CODE>
      <ENT_AMT>122</ENT_AMT>
      <ACCTD_AMT>20.33</ACCTD_AMT>
      <VAT_CODE>VAT22%</VAT_CODE>
    </INVOICE>
  </VENDOR>
  ...
  ...
  ...
  ...
  ...
  ...
  ...
  ...
  ...
  ...
  ...
  ...
  ...
</INVOICELIST>

The report should show each VENDOR and their INVOICE data with a SUMMARY section that appears only on the last page, placed at the bottom of the page. The template for this is shown in the following figure:
Insert a Microsoft Word section break (type: next page) on the first page of the template. For the final page, insert new line characters to position the summary table at the bottom of the page. The summary table is shown in the following figure:

**Last Page Only Layout**

<table>
<thead>
<tr>
<th>Last Page Placeholder</th>
</tr>
</thead>
</table>

**Tax Summary**

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

In this example:

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

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

to declare the last page layout. Any content above or below this statement will appear on the last page only. The content above the statement is regarded as the header and the content below the statement is regarded as the footer.
If your reports contains headers and footers that you want to carry over onto the last page, you must reinsert them on the last page. For more information about headers and footers see Defining Headers and Footers, page 2-16.

You must insert a section break (type: next page) into the document to specify the last page layout. This example is available in the samples folder of the Oracle XML Publisher Template Builder for Word installation.

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

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

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

### End on Even or End on Odd Page

If your report has different odd and even page layouts, you may want to force your report to end specifically on an odd or even page. For example, you may include the terms and conditions of a purchase order in the footer of your report using the different odd/even footer functionality (see Different First Page and Different Odd and Even Page Support, page 2-17) and you want to ensure that the terms and conditions are printed on the final page.

Or, you may have binding requirements to have your report end on an even page, without specific layout.

**To end on an even page with layout:**

Insert the following syntax in a form field in your template:

```xml
<?section:force-page-count;'end-on-even-layout'?>
```

**To end on an odd page layout:**

```xml
<?section:force-page-count;'end-on-odd-layout'?>
```

If you do not have layout requirements for the final page, but would like a blank page ejected to force the page count to the preferred odd or even, use the following syntax:

```xml
<?section:force-page-count;'end-on-even'?>
```

or

```xml
<?section:force-page-count;'end-on-odd'?>
```

### Hyperlinks

XML Publisher supports several different types of hyperlinks. The hyperlinks can be fixed or dynamic and can link to either internal or external destinations. Hyperlinks can also be added to shapes.
• To insert static hyperlinks to either text or a shape, use your word processing application’s insert hyperlink feature:
  1. Select the text or shape.
  2. Use the right-mouse menu to select **Hyperlink**; or, select **Hyperlink** from the **Insert** menu.
  3. Enter the URL using any of the methods provided on the **Insert Hyperlink** dialog box.

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

![Insert Hyperlink Dialog Box](image)

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

```plaintext
(URL_LINK)
```

where URL_LINK is the incoming data element name.

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

```plaintext
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.

![Insert Hyperlink dialog box](image)

- You can also pass parameters at runtime to construct a dynamic URL.

Enter the parameter and element names surrounded by braces to build up the URL as follows:

```
{$SERVER_URL}{REPORT}/cstid={CUSTOMER_ID}
```

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

```
```

### Inserting Internal Links

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

1. Position your cursor in the desired destination in your document.

2. Select **Insert > Bookmark**...

3. In the **Bookmark** dialog, enter a name for this bookmark, and select **Add**.

4. Select the text or shape in your document that you want to link back to the
5. Use the right-mouse menu to select Hyperlink; or select Hyperlink from the Insert menu.

6. On the Insert Hyperlink dialog, select Bookmark.

7. Choose the bookmark you created from the list.

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

### Table of Contents

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

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

To create dynamic headings:

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

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

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

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

### Generating Bookmarks in PDF Output

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

For information on creating the table of contents, see Table of Contents, page 2-58.

- To create links for a static table of contents:

   Enter the syntax:

   ```xml
   <?copy-to-bookmark:?>
   ```

   directly above your table of contents and
To create links for a dynamic table of contents:
Enter the syntax:
<?convert-to-bookmark:?>
directly above the table of contents and
<?end convert-to-bookmark:?>
directly below the table of contents.

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

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

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

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

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

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

   <?population>10000?>

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

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

**Drop Down Lists**

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

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

To create the index for the continent example:

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

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

3. Add each value to the Drop-down item field and the click Add to add it to the Items in drop-down list group. The values will be indexed starting from one for the first, and so on. For example, the list of continents will be stored as follows:

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Asia</td>
</tr>
<tr>
<td>2</td>
<td>North America</td>
</tr>
</tbody>
</table>
### Index Value

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>South America</td>
</tr>
<tr>
<td>4</td>
<td>Europe</td>
</tr>
<tr>
<td>5</td>
<td>Africa</td>
</tr>
<tr>
<td>6</td>
<td>Australia</td>
</tr>
</tbody>
</table>

4. Now use the Help Text box to enter the XML element name that will hold the index for the drop-down field values.

For this example, enter

```xml
<continentIndex/>
```

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

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

The template to create this report is shown in the next figure:

<table>
<thead>
<tr>
<th>Country</th>
<th>Population</th>
<th>more than 10M?</th>
<th>Continent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chad</td>
<td>7,360,000</td>
<td>☐</td>
<td>Africa</td>
</tr>
<tr>
<td>China</td>
<td>1,265,530,000</td>
<td>☐</td>
<td>Asia</td>
</tr>
<tr>
<td>Chile</td>
<td>14,677,000</td>
<td>☑</td>
<td>South America</td>
</tr>
<tr>
<td>Sweden</td>
<td>8,887,000</td>
<td>☐</td>
<td>Europa</td>
</tr>
<tr>
<td>United States</td>
<td>270,312,000</td>
<td>☑</td>
<td>North America</td>
</tr>
<tr>
<td>New Zealand</td>
<td>3,625,000</td>
<td>☐</td>
<td>Australia</td>
</tr>
</tbody>
</table>

The template to create this report is shown in the next figure:

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

Conditional Formatting

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

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

**If Statements**

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

1. Insert the following syntax to designate the beginning of the conditional area.
   
   ```xml
   <?if:condition?>
   ```

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

```xml
<?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

<table>
<thead>
<tr>
<th>Invoice Num</th>
<th>Invoice Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group: Invoices 1234566</td>
<td>1-Jan-2004</td>
</tr>
</tbody>
</table>

<?end if?>

End: Suppliers
```

**If Statements in Boilerplate Text**

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

The program was (not) successful.

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

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

**Note:** For more information on context commands, see Using Context Commands, page 2-125.

For example, if you construct the code as follows:

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

The following undesirable result will occur:

The program was not successful.

because XML Publisher applies the instructions to the block by default. To specify that the if statement should be inserted into the inline sequence, enter the following:

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

This construction will result in the following display:

The program was successful.

If SUCCESS does not equal 'N';

or

The program was not successful.

If SUCCESS equals 'N'.

**If-then-Else Statements**

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

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

You can also nest these statements as follows:

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

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

```xml
<?xdofx:if element_condition then result1 else result2 end if?>
```
For example, the following statement tests the AMOUNT element value. If the value is greater than 1000, show the word "Higher"; if it is less than 1000, show the word "Lower"; if it is equal to 1000, show "Equal":

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

Choose Statements

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

Use the following syntax for these elements:

<?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.
Creating an RTF Template

Column Formatting

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

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

```
<Grp:VAT>
  <Choose>
    <When EXEMPT_FLAG='^'>
      VAT 15%  |  1000  |  1000  |  1000  |  1000  |  1000
    </When>
    <When POSTED_FLAG='^'>
      VAT 15%  |  1000  |  1000  |  1000  |  1000  |  1000
    </When>
    Otherwise
      VAT 15%  |  1000  |  1000  |  1000  |  1000  |  1000
    </Otherwise>
  </Choose>
</Grp:VAT>
```
Notice the type attribute associated with the items element. In this XML it is marked as 'PUBLIC' meaning the list is a public list rather than a 'PRIVATE' list. For the "public" version of the list we do not want to show the quantity column in the output, but we want to develop only one template for both versions based on the list type.

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Quantity</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>grp:ItemPlasma TV</td>
<td>20</td>
<td>1,000.00</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Default Text</th>
<th>Form Field Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grp:Item</td>
<td>&lt;?for-each:item?&gt;</td>
<td>Holds the opening for-each loop for the item element.</td>
</tr>
<tr>
<td>Plasma TV</td>
<td>&lt;?name?&gt;</td>
<td>The placeholder for the name element from the XML file.</td>
</tr>
<tr>
<td>Default Text</td>
<td>Form Field Entry</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>IF</td>
<td>&lt;?if@column:/items/@type=&quot;PRIVATE&quot;?&gt;</td>
<td>The opening of the if statement to test for the attribute value &quot;PRIVATE&quot;. Note that this syntax uses an XPath expression to navigate back to the &quot;items&quot; level of the XML to test the attribute. For more information about using XPath in your templates, see XPath Overview, page 2-121.</td>
</tr>
<tr>
<td>Quantity</td>
<td>N/A</td>
<td>Boilerplate heading</td>
</tr>
<tr>
<td>end-if</td>
<td>&lt;?end if?&gt;</td>
<td>Ends the if statement.</td>
</tr>
<tr>
<td>20</td>
<td>&lt;?if@column:/items/@type=&quot;PRIVATE&quot;?&gt;&lt;?quantity?&gt;&lt;end if?&gt;</td>
<td>The placeholder for the quantity element surrounded by the &quot;if&quot; statement.</td>
</tr>
<tr>
<td>1,000.00</td>
<td>&lt;?price?&gt;</td>
<td>The placeholder for the price element.</td>
</tr>
<tr>
<td>end grp</td>
<td>&lt;?end for-each?&gt;</td>
<td>Closing tag of the for-each loop.</td>
</tr>
</tbody>
</table>

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

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

**Note:** The @column clause is an example of a context command. For more information, see Using Context Commands, page 2-125.

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plasma TV</td>
<td>4,000.00</td>
</tr>
<tr>
<td>DVD Player</td>
<td>300.00</td>
</tr>
<tr>
<td>VCR</td>
<td>200.00</td>
</tr>
<tr>
<td>Receiver</td>
<td>350.00</td>
</tr>
</tbody>
</table>
If the same XML data contained the type attribute set to "PRIVATE" the following output would be rendered from the same template:

<table>
<thead>
<tr>
<th>Name</th>
<th>Quantity</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plasma TV</td>
<td>10</td>
<td>4,000.00</td>
</tr>
<tr>
<td>DVD Player</td>
<td>3</td>
<td>300.00</td>
</tr>
<tr>
<td>VCR</td>
<td>20</td>
<td>200.00</td>
</tr>
<tr>
<td>Receiver</td>
<td>22</td>
<td>350.00</td>
</tr>
</tbody>
</table>

**Row Formatting**

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

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

**Conditionally Displaying a Row**

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

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

Note the following fields from the sample figure:

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

**Conditionally Highlighting a Row**

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

The following table shows values of the form fields in the template:

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

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

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

For this example we will use the following XML:

```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 figure:

<table>
<thead>
<tr>
<th>Account</th>
<th>Debit</th>
<th>Credit</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR:Account1-232-4444</td>
<td>CH1100.00</td>
<td>CH2100.00</td>
</tr>
</tbody>
</table>

The field definitions for the template are shown in the following table:
### Default Text Entry

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

The code to highlight the debit column as shown in the table is:

```
<?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 xdofo: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, #FFFFFF).

The output from this template is displayed in the following figure:
Page-Level Calculations

Displaying Page Totals

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

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

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

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

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

```
<?add-page-total: TotalFieldName; 'element'?>
```

where

- `TotalFieldName` is the name you assign to your total (to reference later) and
- `'element'` is the XML element field to be totaled.

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

Then when you want to display the total field, enter the following syntax:

```
<?show-page-total: TotalFieldName; 'Oracle-number-format'?>
```

where

- `TotalFieldName` is the name you assigned to give the page total field above and
- `Oracle-number-format` is the format you wish to use to for the display, using the Oracle format mask (for example: C9G999D00). For the list of Oracle format mask

<table>
<thead>
<tr>
<th>Account</th>
<th>Debit</th>
<th>Credit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-100-3333</td>
<td>100.00</td>
<td>300.00</td>
</tr>
<tr>
<td>1-101-3533</td>
<td>220.00</td>
<td>30.00</td>
</tr>
<tr>
<td>1-130-3343</td>
<td>240.00</td>
<td>1100.00</td>
</tr>
<tr>
<td>1-153-3033</td>
<td>3000.00</td>
<td>300.00</td>
</tr>
</tbody>
</table>
symbols, see Using the Oracle Format Mask, page 2-112.

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

This example uses the following XML:

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

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

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

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

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

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

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

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

`<?show-page-total:dt;'C9G990D00';'(C9G990D00)'?>`

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;'C9G990D00';'(C9G990D00)'?>`

Page Total Credit: `<?show-page-total:ct;'C9G990D00';'(C9G990D00)'?>`

Page Total Balance: `<?show-page-total:net;'C9G990D00';'(C9G990D00)'?>`

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

<table>
<thead>
<tr>
<th></th>
<th>Debit</th>
<th>Credit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100.00</td>
<td>90.00</td>
</tr>
<tr>
<td></td>
<td>110.00</td>
<td>80.00</td>
</tr>
<tr>
<td></td>
<td>120.00</td>
<td>70.00</td>
</tr>
<tr>
<td></td>
<td>130.00</td>
<td>60.00</td>
</tr>
<tr>
<td></td>
<td>140.00</td>
<td>50.00</td>
</tr>
<tr>
<td></td>
<td>150.00</td>
<td>40.00</td>
</tr>
</tbody>
</table>

Page Total Debit: 750.00
Page Total Credit: 350.00
Page Total Balance: 360.00

**Brought Forward/Carried Forward Totals**

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

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

An example is displayed in the following figure:

<table>
<thead>
<tr>
<th>Inv</th>
<th>Date</th>
<th>Amount</th>
<th>Brought Forward</th>
<th>Carried Forward</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>1-Jan-05</td>
<td>100</td>
<td></td>
<td>300</td>
</tr>
<tr>
<td>1002</td>
<td>1-Jan-05</td>
<td>100</td>
<td></td>
<td>300</td>
</tr>
<tr>
<td>1003</td>
<td>1-Jan-05</td>
<td>100</td>
<td>100</td>
<td>400</td>
</tr>
</tbody>
</table>

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

This functionality is an extension of the Page Totals, page 2-74 feature. The following example walks through the syntax and setup required to display the brought forward and carried forward totals in your published report.

Assume you have the following XML:

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

The following sample template creates the invoice table and declares a placeholder that will hold your page total:
The fields in the template have the following values:

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

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

```xml
<xdofo:inline-total
    display-condition="exceptfirst"
    name="InvAmt">

    Brought Forward:
    <xdofo:show-brought-forward
        name="InvAmt"
        format="99G999G999D00"/>
</xdofo:inline-total>
```

The following table describes the elements comprising the brought forward syntax:
**Code Element** | **Description and Usage**
--- | ---
inline-total | This element has two properties:
- *name* - name of the variable you declared for the field.
- *display-condition* - sets the display condition. This is an optional property that takes one of the following values:
  - *first* - the contents appear only on the first page
  - *last* - the contents appear only on the last page
  - *exceptfirst* - contents appear on all pages except first
  - *exceptlast* - contents appear on all pages except last
  - *everytime* - (default) contents appear on every page

In this example, *display-condition* is set to "exceptfirst" to prevent the value from appearing on the first page where the value would be zero.

**Brought Forward:**

This string is optional and will display as the field name on the report.

**show-brought-forward**

Shows the value on the page. It has the following two properties:
- *name* - the name of the field to show. In this case, "InvAmt". This property is mandatory.
- *format* - the Oracle number format to apply to the value at runtime. This property is optional, but if you want to supply a format mask, you must use the Oracle format mask. For more information, see Using the Oracle Format Mask, page 2-112.

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

If you want the brought forward total to display in the header, you must insert the full code string into the header because Microsoft Word does not support form fields in the header or footer regions. However, you can alternatively use the start body/end body syntax which allows you to define what the body area of the report will be. XML Publisher will recognize any content above the defined body area as header content, and any content below as the footer. This allows you to use form fields. See Multiple or
Complex Headers and Footers, page 2-16 for details.

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

```xml
<xdof:inline-total
  display-condition="exceptlast"
  name="InvAmt">
  Carried Forward:
  <xdof:show-carry-forward
    name="InvAmt"
    format="99G999G999D00"/>
</xdof:inline-total>
```

Note the following differences with the brought-forward object:

- The `display-condition` is set to `exceptlast` so that the carried forward total will display on every page except the last page.

- The display string is "Carried Forward".

- The `show-carry-forward` element is used to show the carried forward value. It has the same properties as `brought-carried-forward`, described above.

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

Running Totals

**Example**

The variable functionality (see Using Variables, page 2-89) can be used to add a running total to your invoice listing report. This example assumes the following XML structure:

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

Using this XML, we want to create the report that contains running totals as shown in the following figure:
To create the Running Total field, define a variable to track the total and initialize it to 0. The template is shown in the following figure:

<table>
<thead>
<tr>
<th>Invoice Number</th>
<th>Invoice Date</th>
<th>Amount</th>
<th>Running Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000-1</td>
<td>1-Jan-2005</td>
<td>100.00</td>
<td>100.00</td>
</tr>
<tr>
<td>1000-2</td>
<td>10-Jan-2005</td>
<td>200.00</td>
<td>300.00</td>
</tr>
<tr>
<td>1000-3</td>
<td>11-Jan-2005</td>
<td>150.00</td>
<td>450.00</td>
</tr>
</tbody>
</table>

To create the Running Total field, define a variable to track the total and initialize it to 0. The template is shown in the following figure:

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

Sorting

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

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

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

```xml
<?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:

```xml
```

Checking for Nulls

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

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

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

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

- To define behavior when the element is present, but is null, use the following:
  ```xml
  <?if:element_name and element_name=">desired behavior <?end if?>
  ```

- To define behavior when the element is not present, use the following:
  ```xml
  <?if:not(element_name)>desired behavior <?end if?>
  ```
Regrouping the XML Data

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

XML Sample

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

```xml
<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 Tyler</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>
</CATALOG>
```

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:

```xml
<?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:

<table>
<thead>
<tr>
<th>Group by Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>Country USA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group by Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year 2000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Title</th>
<th>Artist</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group Details My CD</td>
<td>John Doe</td>
<td>1.00 End Group</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Country: USA</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Year: 1985</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Title</th>
<th>Artist</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empire Burlesque</td>
<td>Bob Dylan</td>
<td>10.90</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Country: UK</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Year: 1988</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Title</th>
<th>Artist</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hide your heart</td>
<td>Bonnie Tylor</td>
<td>9.90</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Year: 1990</th>
<th></th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Title</th>
<th>Artist</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Still got the blues</td>
<td>Gary More</td>
<td>10.20</td>
</tr>
<tr>
<td>This is US</td>
<td>Gary Lee</td>
<td>12.20</td>
</tr>
</tbody>
</table>

**Regrouping by an Expression**

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

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

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

**Example**

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

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

```
<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:
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:

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

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

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

Note the following about the form field tags:

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

2-89

(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.

Variables, Parameters, and Properties

Using Variables

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

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

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

<?xdoxslt:set_variable($_XDOCTX, 'variable name', value)?>

Use the following syntax to retrieve a variable value:

<?xdoxslt:get_variable($_XDOCTX, 'variable name')?>

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

<?xdoxslt:set_variable($_XDOCTX, 'x', xdoxslt:get_variable($_XDOCTX, 'x') + 1)?>

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

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

Creating an RTF Template 2-89
See the section on Running Totals, page 2-80 for an example of the usage of updateable variables.

**Defining Parameters**

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

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

**Using a parameter in a template**

1. Declare the parameter in the template.

   Use the following syntax to declare the parameter:
   
   ```xml
   <xsl:param name="PARAMETERNAME" select="DEFAULT" xdofo:ctx="begin"/>
   ```

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

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

2. Refer to the parameter in the template by prefixing the name with a "$" character.

   For example, if you declare the parameter name to be "InvThresh", then reference the value using "$InvThresh".

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

   Prior to calling either the FOProcessor API (Core) or the TemplateHelper API (E-Business Suite) create a Properties class and assign a property to it for the parameter value as follows:
   
   ```java
   Properties prop = new Properties();
   prop.put("xslt.InvThresh", "1000");
   ```

   For more information, see Calling XML Publisher APIs, *Oracle XML Publisher Administration and Developer’s Guide*.

**Example: Passing an invoice threshold parameter**

This example illustrates how to declare a parameter in your template that will filter your data based on the value of the parameter.
The following XML sample lists invoice data:

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

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

---

### InvThresh Declaration

<table>
<thead>
<tr>
<th>Invoice Number</th>
<th>Invoice Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>13222-2</td>
<td>$100.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Form Field Help Text Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InvThreshDeclaration</td>
<td>&lt;xsl:param name=&quot;InvThresh&quot; xdofo:ctx=&quot;begin&quot;/&gt;</td>
<td>Declares the parameter InvThresh.</td>
</tr>
<tr>
<td>FE</td>
<td>&lt;?for-each:INVOICE?&gt;</td>
<td>Begins the repeating group for the INVOICE element.</td>
</tr>
<tr>
<td>IF</td>
<td>&lt;?if:AMOUNT&gt;$InvThresh?&gt;</td>
<td>Tests the value of the AMOUNT element to determine if it is greater than the value of InvThresh.</td>
</tr>
<tr>
<td>$100.00</td>
<td>&lt;?AMOUNT?&gt;</td>
<td>Placeholder for the AMOUNT element.</td>
</tr>
<tr>
<td>EI</td>
<td>&lt;?end if?&gt;</td>
<td>Closing tag for the if statement.</td>
</tr>
</tbody>
</table>
In this template, only INVOICE elements with an AMOUNT greater than the InvThresh parameter value will be displayed. If we pass in a parameter value of 1,000, the following output shown in the following figure will result:

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

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

Setting Properties

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

Note: See XML Publisher Configuration File, Oracle XML Publisher Administration and Developer’s Guide for more information about the XML Publisher Configuration file and the available properties.

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

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

Name - enter the XML Publisher property name prefixed with "xdo-"

Type - select "Text"

Value - enter the property value. To reference an element from the incoming XML data, enter the path to the XML element enclosed by curly braces. For example:{/root/password}
Embedding a Font Reference
For this example, suppose you want to use a font in the template called "XMLPScript". This font is not available as a regular font on your server, therefore you must tell XML Publisher where to find the font at runtime. You tell XML Publisher where to find the font by setting the "font" property. Assume the font is located in "/tmp/fonts", then you would enter the following in the Properties dialog:

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

**Type:** Text

**Value:** {/root/font}

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

For more information about setting font properties, see Font Definitions, *Oracle XML Publisher Administration and Developer’s Guide*. 

---

*Creating an RTF Template 2-93*
Securing a PDF Output
For this example, suppose you want to use a password from the XML data to secure the PDF output document. The XML data is as follows:

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

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

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

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

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

For example, you could set up the following parameters:

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

You would then enter the following in the Properties dialog:

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

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

For more information about template parameters, see Defining Parameters in Your Template, page 2-90.
Advanced Report Layouts

Batch Reports

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

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

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

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

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

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

```
<?for-each@section:group name?>
```

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

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

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

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

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

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

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

**Cross-Tab Support**

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

This example uses the following XML sample:

```xml
<ROWSET>
  <RESULTS>
    <INDUSTRY>Motor Vehicle Dealers</INDUSTRY>
    <YEAR>2005</YEAR>
    <QUARTER>Q1</QUARTER>
    <SALES>1000</SALES>
  </RESULTS>
  
  <RESULTS>
    <INDUSTRY>Motor Vehicle Dealers</INDUSTRY>
    <YEAR>2005</YEAR>
    <QUARTER>Q2</QUARTER>
    <SALES>2000</SALES>
  </RESULTS>
  
  <RESULTS>
    <INDUSTRY>Motor Vehicle Dealers</INDUSTRY>
    <YEAR>2004</YEAR>
    <QUARTER>Q1</QUARTER>
    <SALES>3000</SALES>
  </RESULTS>
  
  <RESULTS>
    <INDUSTRY>Motor Vehicle Dealers</INDUSTRY>
    <YEAR>2004</YEAR>
    <QUARTER>Q2</QUARTER>
    <SALES>3000</SALES>
  </RESULTS>
  
  <RESULTS>
    <INDUSTRY>Motor Vehicle Dealers</INDUSTRY>
    <YEAR>2003</YEAR>
    ...  
  </RESULTS>
  
  <RESULTS>
    <INDUSTRY>Home Furnishings</INDUSTRY>
    ...  
  </RESULTS>
  
  <RESULTS>
    <INDUSTRY>Electronics</INDUSTRY>
    ...  
  </RESULTS>
  
  <RESULTS>
    <INDUSTRY>Food and Beverage</INDUSTRY>
    ...  
  </RESULTS>

</ROWSET>

From this XML we will generate a report that shows each industry and totals the sales by year as shown in the following figure:
The template to generate this report is shown in the following figure. The form field entries are shown in the subsequent table.

<table>
<thead>
<tr>
<th>Default Text Entry</th>
<th>Form Field Help Text</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>header column</td>
<td>&lt;?horizontal-break-table;1?&gt;</td>
<td>Defines the first column as a header that should repeat if the table breaks across pages. For more information about this syntax, see Defining Columns to Repeat Across Pages, page 2-100.</td>
</tr>
<tr>
<td>for:</td>
<td>&lt;?for-each-group@column:RESULTS;YEAR?&gt;</td>
<td>Uses the regrouping syntax (see Regrouping the XML Data, page 2-83) to group the data by YEAR; and the @column context command to create a table column for each group (YEAR). For more information about context commands, see Using the Context Commands, page 2-125.</td>
</tr>
<tr>
<td>YEAR</td>
<td>&lt;?YEAR?&gt;</td>
<td>Placeholder for the YEAR element.</td>
</tr>
<tr>
<td>end</td>
<td>&lt;?end for-each-group?&gt;</td>
<td>Closes the for-each-group loop.</td>
</tr>
<tr>
<td>for:</td>
<td>&lt;?for-each-group:RESULTS;INDUSTRY?&gt;</td>
<td>Begins the group to create a table row for each INDUSTRY.</td>
</tr>
<tr>
<td>INDUSTRY</td>
<td>&lt;?INDUSTRY?&gt;</td>
<td>Placeholder for the INDUSTRY element.</td>
</tr>
</tbody>
</table>

The template to generate this report is shown in the following figure. The form field entries are shown in the subsequent table.
### Default Text Entry Form Field Help Text Description

<table>
<thead>
<tr>
<th>Default Text Entry</th>
<th>Form Field Help Text</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>for:</td>
<td>&lt;?for-each-group@cell:current-group();YEAR?&gt;</td>
<td>Uses the regrouping syntax (see Regrouping the XML Data, page 2-83) to group the data by YEAR; and the <code>@cell</code> context command to create a table cell for each group (YEAR).</td>
</tr>
<tr>
<td>sum(Sales)</td>
<td>&lt;?sum(current-group()//SALE S)?&gt;</td>
<td>Sums the sales for the current group (YEAR).</td>
</tr>
<tr>
<td>end</td>
<td>&lt;?end for-each-group?&gt;</td>
<td>Closes the for-each-group statement.</td>
</tr>
<tr>
<td>end</td>
<td>&lt;?end for-each-group?&gt;</td>
<td>Closes the for-each-group statement.</td>
</tr>
</tbody>
</table>

Note that only the first row uses the `@column` context to determine the number of columns for the table. All remaining rows need to use the `@cell` context to create the table cells for the column. (For more information about context commands, see Using the Context Commands, page 2-125.)

### 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?>`  
  `<?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:

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

**Defining Columns to Repeat Across Pages**

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

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

where number is the number of columns (starting from the left) to repeat.

Note that this functionality is supported for PDF output only.

**Example of Dynamic Data Columns**

A template is required to display test score ranges for school exams. Logically, you want the report to be arranged as shown in the following table:
but you do not know how many Test Score Ranges will be reported. The number of Test Score Range columns is dynamic, depending on the data.

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

```xml
<?xml version="1.0" encoding="utf-8"?>
<TestScoreTable>
  <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 2-9 for more information on using form fields).
• **Test Score** is the boilerplate column heading.

• Test Category is the placeholder for the `<TestCategory>` data element, that is, "Mathematics," which will also be the row heading.

• The second column is the one to be split dynamically. The width you specify will be divided by the number of columns of data. In this case, there are 5 data columns.

• The second column will contain the dynamic "range" data. The width of the column will be divided according to the split column width. Because this example does not contain the unit value tag (`<?split-column-width-unit:value?>`), the column will be split on a percentage basis. Wrapping of the data will occur if required.

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

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

<table>
<thead>
<tr>
<th>Test Score</th>
<th>0.20</th>
<th>21.40</th>
<th>41.60</th>
<th>61.80</th>
<th>81.100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mathematics</td>
<td>30</td>
<td>45</td>
<td>50</td>
<td>102</td>
<td>22</td>
</tr>
</tbody>
</table>

Number and Date Formatting

Number Formatting

XML Publisher supports two methods for specifying the number format:

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

**Note:** You can also use the native XSL format-number function to format numbers. See: Native XSL Number Formatting, page 2-129.

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

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

Data Source Requirements

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

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

Translation Considerations

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

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

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

Using the Microsoft Number Format Mask

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

To apply a number format to a form field:

1. Open the Form Field Options dialog box for the placeholder field.

2. Set the Type to Number.

3. Select the appropriate Number format from the list of options.

Supported Microsoft Format Mask Definitions

The following table lists the supported Microsoft format mask definitions:
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Location</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Number</td>
<td>Digit. Each explicitly set 0 will appear, if no other number occupies the position.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Format mask: 00.0000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data: 1.234</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Display: 01.2340</td>
</tr>
<tr>
<td>#</td>
<td>Number</td>
<td>Digit. When set to #, only the incoming data is displayed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Format mask: ##.####</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data: 1.234</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Display: 1.234</td>
</tr>
<tr>
<td>.</td>
<td>Number</td>
<td>Determines the position of the decimal separator. The decimal separator symbol used will be determined at runtime based on template locale.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Format mask: #,##0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data: 1234.56</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Display for English locale: 1,234.56</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Display for German locale: 1.234,56</td>
</tr>
<tr>
<td>-</td>
<td>Number</td>
<td>Determines placement of minus sign for negative numbers.</td>
</tr>
<tr>
<td>,</td>
<td>Number</td>
<td>Determines the placement of the grouping separator. The grouping separator symbol used will be determined at runtime based on template locale.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Format mask: #,##0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data: 1234.56</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Display for English locale: 1,234.56</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Display for German locale: 1.234,56</td>
</tr>
</tbody>
</table>
### Symbol Location Meaning

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Location</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>Number</td>
<td>Separates mantissa and exponent in a scientific notation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.###E+0 plus sign always shown for positive numbers</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.###E-0 plus sign not shown for positive numbers</td>
</tr>
<tr>
<td>;</td>
<td>Subpattern boundary</td>
<td>Separates positive and negative subpatterns. See Note below.</td>
</tr>
<tr>
<td>%</td>
<td>Prefix or Suffix</td>
<td>Multiply by 100 and show as percentage</td>
</tr>
<tr>
<td>'</td>
<td>Prefix or Suffix</td>
<td>Used to quote special characters in a prefix or suffix.</td>
</tr>
</tbody>
</table>

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

### Using the Oracle Format Mask

To apply the Oracle format mask to a form field:

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

   where

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

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

2-106  Oracle XML Publisher Report Designer's Guide
The following table lists the supported Oracle number format mask symbols and their definitions:

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

**Date Formatting**

XML Publisher supports three methods for specifying the date format:

- Specify an explicit date format mask using Microsoft Word’s native date format mask.
• Specify an explicit date format mask using Oracle's format-date function.

• Specify an abstract date format mask using Oracle's abstract date format masks. (Recommended for multilingual templates.)

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

Data Source Requirements

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

YYYY-MM-DDThh:mm:ss+HH:MM

where

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

An example of this construction is:

2005-01-01T09:30:10-07:00

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

Translation Considerations

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

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

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

Using the Microsoft Date Format Mask

To apply a date format to a form field:

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

If you do not specify the mask in the Date format field, the abstract format mask "MEDIUM" will be used as default. See Oracle Abstract Format Masks, page 2-116 for the description.

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

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

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

Using the Oracle Format Mask

To apply the Oracle format mask to a date field:

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

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

   or

   ```xml
   <?format-date-and-calendar:date_string;
   'ABSTRACT_FORMAT_MASK';'CALENDAR_NAME';'TIMEZONE'>
   ```
where time zone is optional. The detailed usage of format mask, calendar and time zone is described below.

If no format mask is specified, the abstract format mask "MEDIUM" will be used as default.

Example form field help text entry:

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

The following table lists the supported Oracle format mask components:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Punctuation and quoted text are reproduced in the result.</td>
</tr>
<tr>
<td>/</td>
<td>AD indicator with or without periods.</td>
</tr>
<tr>
<td>,</td>
<td>A.D.</td>
</tr>
<tr>
<td>,</td>
<td>AM</td>
</tr>
<tr>
<td>.</td>
<td>Meridian indicator with or without periods.</td>
</tr>
<tr>
<td>;</td>
<td>A.M.</td>
</tr>
<tr>
<td>:</td>
<td>BC</td>
</tr>
<tr>
<td>&quot;text&quot;</td>
<td>BC indicator with or without periods.</td>
</tr>
<tr>
<td>AD</td>
<td>Century. For example, 2002 returns 21; 2000 returns 20.</td>
</tr>
<tr>
<td>A.D.</td>
<td>DAY</td>
</tr>
<tr>
<td>AM</td>
<td>Name of day, padded with blanks to length of 9 characters.</td>
</tr>
<tr>
<td>A.M.</td>
<td>D</td>
</tr>
<tr>
<td>BC</td>
<td>Day of week (1-7).</td>
</tr>
<tr>
<td>B.C.</td>
<td>DD</td>
</tr>
<tr>
<td>CC</td>
<td>Day of month (1-31).</td>
</tr>
<tr>
<td>D</td>
<td>DDD</td>
</tr>
<tr>
<td>DAY</td>
<td>Day of year (1-366).</td>
</tr>
<tr>
<td>Symbol</td>
<td>Meaning</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>DL</td>
<td>Returns a value in the long date format.</td>
</tr>
<tr>
<td>DS</td>
<td>Returns a value in the short date format.</td>
</tr>
<tr>
<td>DY</td>
<td>Abbreviated name of day.</td>
</tr>
<tr>
<td>E</td>
<td>Abbreviated era name.</td>
</tr>
<tr>
<td>EE</td>
<td>Full era name.</td>
</tr>
<tr>
<td>FF[1..9]</td>
<td>Fractional seconds. Use the numbers 1 to 9 after FF to specify the number of digits in the fractional second portion of the datetime value returned.</td>
</tr>
<tr>
<td></td>
<td>Example: 'HH:MI:SS,FF3'</td>
</tr>
<tr>
<td>HH</td>
<td>Hour of day (1-12).</td>
</tr>
<tr>
<td>HH12</td>
<td>Hour of day (1-12).</td>
</tr>
<tr>
<td>HH24</td>
<td>Hour of day (0-23).</td>
</tr>
<tr>
<td>MI</td>
<td>Minute (0-59).</td>
</tr>
<tr>
<td>MM</td>
<td>Month (01-12; JAN = 01).</td>
</tr>
<tr>
<td>MON</td>
<td>Abbreviated name of month.</td>
</tr>
<tr>
<td>MONTH</td>
<td>Name of month, padded with blanks to length of 9 characters.</td>
</tr>
<tr>
<td>PM</td>
<td>Meridian indicator with or without periods.</td>
</tr>
<tr>
<td>P.M.</td>
<td></td>
</tr>
<tr>
<td>RR</td>
<td>Lets you store 20th century dates in the 21st century using only two digits.</td>
</tr>
<tr>
<td>RRRR</td>
<td>Round year. Accepts either 4-digit or 2-digit input. If 2-digit, provides the same return as RR. If you don't want this functionality, then simply enter the 4-digit year.</td>
</tr>
<tr>
<td>SS</td>
<td>Seconds (0-59).</td>
</tr>
<tr>
<td>Symbol</td>
<td>Meaning</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
</tr>
</tbody>
</table>
| TZD    | Daylight savings information. The TZD value is an abbreviated time zone string with daylight savings information. It must correspond to the region specified in TZR.  
Example:  
PST (for Pacific Standard Time)  
PDT (for Pacific Daylight Time) |
| TZH    | Time zone hour. (See TZM format element.) |
| TZM    | Time zone minute. (See TZH format element.)  
Example:  
‘HH:MI:SS.FFTZH:TZM’ |
| TZR    | Time zone region information. The value must be one of the time zone regions supported in the database. Example: PST (Pacific Standard Time) |
| WW     | Week of year (1-53) where week 1 starts on the first day of the year and continues to the seventh day of the year. |
| W      | Week of month (1-5) where week 1 starts on the first day of the month and ends on the seventh. |
| X      | Local radix character. |
| YYYY   | 4-digit year. |
| YY     | Last 2, or 1 digit(s) of year. |
| Y      | |

**Default Format Mask**

If you do not want to specify a format mask with either the MS method or the Oracle method, you can omit the mask definition and use the default format mask. The default format mask is the MEDIUM abstract format mask from Oracle. (See Oracle Abstract Format Masks, page 2-116 for the definition.)

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

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

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

Specify the abstract mask using the following syntax:

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

where fieldname is the XML element tag and

MASK is the Oracle abstract format mask name

For example:

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

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

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

Currency Formatting

XML Publisher enables you to define specific currency format masks to apply to your published data at runtime.
To utilize currency formatting, you must:

1. Define your currency formats in XML Publisher’s Administration interface. See Defining Currency Formats, Oracle XML Publisher Administration and Developer’s Guide.

2. Assign the Currency Format Set as a configuration property at the desired level (site, data definition, or template). It is available from the FO Processing Properties, Oracle XML Publisher Administration and Developer’s Guide list.

3. Enter the format-currency command in your RTF template to apply the format to the field at runtime.

   **To use the format-currency command:**

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

   ```xml
   <?format-currency:ELEMENT_NAME;'currency-format-code'?>
   ```

   For example, to format an element from your XML file named CURRENT_BALANCE according to a currency code USD, enter the following in the form field for the element:

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

   The currency code must correspond to a currency format that is defined in the Currency Format Set to be used with this report. The Currency Format Set can be specified at the site level, data definition level, or template level. For more information, see Defining Currency Formats, Oracle XML Publisher Administration and Developer’s Guide.

---

**Calendar and Time Zone Support**

**Calendar Specification**

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

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

Use one of the following methods to set the calendar type:
• Call the format-date-and-calendar function and declare the calendar type.

    For example:
    <?format-date-and-calendar:hiredate;'LONG_TIME_TZ';'ROC_OFFICIAL'?>

    The following graphic shows the output generated using this definition with locale
    set to zh-TW and time zone set to Asia/Taipei:

        中華民國88年12月31日 星期五 下午 2:15 台北

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

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

Time Zone Specification

There are two ways to specify time zone information:

• Call the format-date or format-date-and-calendar function with the Oracle format.

• Set the user profile option Client Timezone (CLIENT_TIMEZONE_ID) in Oracle
  Applications.

If no time zone is specified, UTC is used.

In the template, the time zone must be specified as a Java time zone string, for example,
America/Los Angeles. The following example shows the syntax to enter in the help text
field of your template:

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

Using External Fonts

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

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

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

\[
\begin{array}{cccc}
\text{Normal} & \text{GnuMICR} & \text{12} & \text{B} \text{ I} \text{ U} \\
\end{array}
\]

formfield method

\[
\begin{array}{cccc}
\text{6} \text{5} \text{4} \text{5} \text{7} \text{5} \text{4} \text{?} \text{4} \text{?} \text{4} \text{?} \\
\end{array}
\]
text method

2. Upload the font to the database using the Font Files subtab of the Administration tab. See Uploading Font Files, Oracle XML Publisher Administration and Developer’s Guide.

3. Create a font mapping set.
   - From the Administration tab, select the Font Mappings subtab.
   - Follow the guidelines for creating a Font Mapping Set, Oracle XML Publisher Administration and Developer’s Guide.

4. Create the font mapping. See Creating a Font Mapping, Oracle XML Publisher Administration and Developer’s Guide.

   Note: Font Family is the exact font name used in Microsoft Word under the Format > Font menu. If you do not use the same name the font will not be picked up at runtime. Style and weight must also match how you use the font in Word. Normal and Normal are typical defaults.

5. Make your font mapping available at the site, template, or data definition level.

   Use the appropriate Configuration page for the desired level. Expand the FO Processing properties group and use the LOV for the Font mapping set property to select the font mapping you want to make available for this level.

   If you add the font to an individual template then only that template can use that font. If you add it to a data definition, then all templates associated with that definition can use the font.

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

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

**Advanced Barcode Formatting**

XML Publisher offers the ability to execute preprocessing on your data prior to applying a barcode font to the data in the output document. For example, you may need to calculate checksum values or start and end bits for the data before formatting them.

The solution requires that you register a barcode encoding class with XML Publisher that can then be instantiated at runtime to carry out the formatting in the template. This is covered in Advanced Barcode Font Formatting Class Implementation, Oracle XML Publisher Administration and Developer’s Guide.

To enable the formatting feature in your template, you must use two commands in your template. The first command registers the barcode encoding class with XML Publisher. This must be declared somewhere in the template prior to the encoding command. The second is the encoding command to identify the data to be formatted.

**Register the Barcode Encoding Class**

Use the following syntax in a form field in your template to register the barcode encoding class:

```xml
<?register-barcode-vendor:java_class_name;barcode_vendor_id?>
```

This command requires a Java class name (this will carry out the encoding) and a barcode vendor ID as defined by the class. This command must be placed in the template before the commands to encode the data in the template. For example:

```xml
```

where

`oracle.apps.xdo.template.rtf.util.barcoder.BarcodeUtil` is the Java class and

`XMLPBarVendor` is the vendor ID that is defined by the class.

**Encode the Data**

To format the data, use the following syntax in a form field in your template:

```xml
<?format-barcode:data;'barcode_type';'barcode_vendor_id'?>
```
where
data is the element from your XML data source to be encoded. For example:
LABEL_ID

barcode_type is the method in the encoding Java class used to format the data (for example: Code128a).

barcode_vendor_id is the ID defined in the register-barcode-vendor field of the first command you used to register the encoding class.

For example:
<?format-barcode:LABEL_ID;'Code128a';'XMLPBarVendor'?>

At runtime, the barcode_type method is called to format the data value and the barcode font will then be applied to the data in the final output.

Advanced Design Options

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

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

XPath Overview

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

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

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

• root

• element

• attribute

• text
Many of these elements are shown in the following sample XML, which contains a catalog of CDs:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<! -  My CD Listing  - >
<CATALOG>
  <CD cattype=Folk>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
  </CD>
  <CD cattype=Rock>
    <TITLE>Hide Your Heart</TITLE>
    <ARTIST>Bonnie Tylor</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <PRICE>9.90</PRICE>
    <YEAR>1988</YEAR>
  </CD>
</CATALOG>
```

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

Locating Data

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

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

```
//CATALOG/CD
```

where

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

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

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

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

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

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

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

```
/CATALOG/CD[PRICE]
```

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

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

This returns the following data from the sample XML document:

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

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

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

The sample returns the first CD element:

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

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

```
/CATALOG/*
```

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

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

/CATALOG/CD[YEAR >=1985]

Starting Reference

The first character in an XPath expression determines the point at which it should start in the XML tree. Statements beginning with a forward slash (/) are considered absolute. No slash indicates a relative reference. An example of a relative reference is:

CD/*

This statement begins the search at the current reference point. That means if the example occurred within a group of statements the reference point left by the previous statement would be utilized.

A noted earlier, double forward slashes (//) retrieve every matching element regardless of location in the document.

Context and Parent

To select current and parent elements, XPath recognizes the dot notation commonly used to navigate directories. Use a single period (.) to select the current node and use double periods (..) to return the parent of the current node. For example, to retrieve all child nodes of the parent of the current node, use:

../*

Therefore, to access all CDs from the sample XML, use the following expression:

/CATALOG/CD/..

You could also access all the CD titles released in 1988 using the following:

/CATALOG/CD/TITLE[../YEAR=1988]

The ../ is used to navigate up the tree of elements to find the YEAR element at the same level as the TITLE, where it is then tested for a match against "1988". You could also use // in this case, but if the element YEAR is used elsewhere in the XML document, you may get erroneous results.

XPath is an extremely powerful standard when combined with RTF templates allowing you to use conditional formatting and filtering in your template.

Namespace Support

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

<?namespace:namespace name= namespace url?>

For example:

Once declared, you can use the namespace in the placeholder markup, for example:

```xml
<?fsg:ReportName?>
```

## Using the Context Commands

The XML Publisher syntax is simplified XSL instructions. This syntax, along with any native XSL commands you may use in your template, is converted to XSL-FO when you upload the template to the Template Manager. The placement of these instructions within the converted stylesheet determines the behavior of your template.

XML Publisher’s RTF processor places these instructions within the XSL-FO stylesheet according to the most common context. However, sometimes you need to define the context of the instructions differently to create a specific behavior. To support this requirement, XML Publisher provides a set of context commands that allow you to define the context (or placement) of the processing instructions. For example, using context commands, you can:

- Specify an if statement in a table to refer to a cell, a row, a column or the whole table.
- Specify a for-each loop to repeat either the current data or the complete section (to create new headers and footers and restart the page numbering)
- Define a variable in the current loop or at the beginning of the document.

You can specify a context for both processing commands using the XML Publisher syntax and those using native XSL.

- To specify a context for a processing command using the simplified XML Publisher syntax, simply add `@context` to the syntax instruction. For example:
  - ```xml
    <?for-each@section:INVOICE?>
    ```
    - specifies that the group INVOICE should begin a new section for each occurrence. By adding the section context, you can reset the header and footer and page numbering.
  - ```xml
    <?if@column:VAT?>
    ```
    - specifies that the if statement should apply to the VAT column only.

- To specify a context for an XSL command, add the `xdofo:ctx="context"` attribute to your tags to specify the context for the insertion of the instructions. The value of the context determines where your code is placed.

  For example:

  ```xml
  <xsl:for-each xdofo:ctx="section" select ="INVOICE">
  <xsl:attribute xdofo:ctx="inblock"
  name="background-color">red</xsl:attribute>
  </xsl:for-each>
  ```

XML Publisher supports the following context types:
The following table shows the default context for the XML Publisher commands:

<table>
<thead>
<tr>
<th>Context</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>section</td>
<td>The statement affects the whole section including the header and footer. For example, a <code>for-each@section</code> context command creates a new section for each occurrence - with restarted page numbering and header and footer. See Batch Reports, page 2-95 for an example.</td>
</tr>
<tr>
<td>column</td>
<td>The statement will affect the whole column of a table. This context is typically used to show and hide table columns depending on the data. See Column Formatting, page 2-67 for an example.</td>
</tr>
<tr>
<td>cell</td>
<td>The statement will affect the cell of a table. This is often used together with <code>@column</code> in cross-tab tables to create a dynamic number of columns. See Cross-Tab Support, page 2-96 for an example.</td>
</tr>
<tr>
<td>block</td>
<td>The statement will affect multiple complete fo:blocks (RTF paragraphs). This context is typically used for if and for-each statements. It can also be used to apply formatting to a paragraph or a table cell. See Cell Highlighting, page 2-72 for an example.</td>
</tr>
<tr>
<td>inline</td>
<td>The context will become the single statement inside an fo:inline block. This context is used for variables.</td>
</tr>
<tr>
<td>incontext</td>
<td>The statement is inserted immediately after the surrounding statement. This is the default for <code>&lt;sort&gt;</code> statements that need to follow the surrounding <code>for-each</code> as the first element.</td>
</tr>
<tr>
<td>inblock</td>
<td>The statement becomes a single statement inside an fo:block (RTF paragraph). This is typically not useful for control statements (such as <code>if</code> and <code>for-each</code>) but is useful for statements that generate text, such as <code>call-template</code>.</td>
</tr>
<tr>
<td>inlines</td>
<td>The statement will affect multiple complete inline sections. An inline section is text that uses the same formatting, such as a group of words rendered as bold. See If Statements in Boilerplate Text, page 2-64.</td>
</tr>
<tr>
<td>begin</td>
<td>The statement will be placed at the beginning of the XSL stylesheet. This is required for global variables. See Defining Parameters, page 2-90.</td>
</tr>
<tr>
<td>end</td>
<td>The statement will be placed at the end of the XSL stylesheet.</td>
</tr>
</tbody>
</table>
### Command Context

<table>
<thead>
<tr>
<th>Command</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply-template</td>
<td>inline</td>
</tr>
<tr>
<td>attribute</td>
<td>inline</td>
</tr>
<tr>
<td>call-template</td>
<td>inblock</td>
</tr>
<tr>
<td>choose</td>
<td>block</td>
</tr>
<tr>
<td>for-each</td>
<td>block</td>
</tr>
<tr>
<td>if</td>
<td>block</td>
</tr>
<tr>
<td>import</td>
<td>begin</td>
</tr>
<tr>
<td>param</td>
<td>begin</td>
</tr>
<tr>
<td>sort</td>
<td>incontext</td>
</tr>
<tr>
<td>template</td>
<td>end</td>
</tr>
<tr>
<td>value-of</td>
<td>inline</td>
</tr>
<tr>
<td>variable</td>
<td>end</td>
</tr>
</tbody>
</table>

### Using XSL Elements

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

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

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

### Apply a Template Rule

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

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

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

**Copy the Current Node**

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

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

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

**Call Template**

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

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

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

**Template Declaration**

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

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

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

**Variable Declaration**

Use this element to declare a local or global variable.

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

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

**Example:**

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

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

**Import Stylesheet**

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

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

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

**XML Publisher Tag:** `<?import:url?>`
Define the Root Element of the Stylesheet

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

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

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

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

**Note:** The namespace must be declared in the template. See Namespace Support, page 2-124.

Native XSL Number Formatting

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

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

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

The full list of FO elements supported by XML Publisher can be found in the Appendix: Supported XSL-FO Elements, page A-1.

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:

```xml
<startbody>

Group: Suppliers, Sort by Supplier

Supplier: Supplier 1

<table>
<thead>
<tr>
<th>Invoice Num</th>
<th>Invoice Date</th>
<th>GL Date</th>
<th>Curr</th>
<th>Entered Amt</th>
<th>Accounted Amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group Invoices</td>
<td>1-1-Jan-2004</td>
<td>1-1-Jan-2004</td>
<td>USD</td>
<td>1000.00</td>
<td>1000.00</td>
</tr>
<tr>
<td>1234566</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Total for Supplier: Supplier 1 1000.00 1000.00

End Suppliers
<endbody>
```

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

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

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

It is likely that you will design several templates that contain common components or functionality, such as address formatting or string formatting functions. Rather than replicate this functionality in every template, you can place the common components in a separate template that can be called at runtime by the other templates. The template containing the common components is a "subtemplate."

If your template requires complex calculations or conditional formatting of tabular data, you can also use a subtemplate for these instructions to keep the primary template simple for your users to understand.

Using a subtemplate consists of four steps:

1. Create the RTF file containing the common components or processing instructions. This is your subtemplate.

2. Register the subtemplate in the Template Manager.

3. Enter the \texttt{import} syntax in the calling or "primary" template to import the subtemplate.

4. Enter the \texttt{call-template} syntax in the primary template in the position that you want the subtemplate contents to render.

Tip: You can use the XML Publisher Desktop Template Viewer to test your template plus subtemplate before loading them to the Template Manager. To do so, you must alter the \texttt{import} template syntax to point to the location of the subtemplate in your local environment. See Testing Your Subtemplate, page 2-133.

Create the Subtemplate RTF File

Enter the components or instructions in an RTF file. To define this file as a subtemplate, enclose the contents in the following tags:

\begin{verbatim}
<template:template_name?>
<end template>
\end{verbatim}

where

\begin{itemize}
  \item \texttt{template_name} is the name you choose for the subtemplate.
\end{itemize}

Note that in a single file, you can have multiple \texttt{<template:template_name?>}\texttt{<end template?>} entries, to mark different "subtemplates" or segments you would like to include in other files.

Register the Subtemplate RTF File in the Template Manager

Register the subtemplate RTF file in the Template Manager. To define it as a subtemplate, select \texttt{Subtemplate = YES}. 
For more information about the Template Manager, see Creating the Template, *Oracle XML Publisher Administration and Developer’s Guide*.

**Enter the Import Syntax in the Primary Template**

Import the subtemplate to the primary template by entering the following syntax in the primary template. The import syntax tells the XML Publisher engine where to find the subtemplate RTF in the Template Manager. This syntax may be entered anywhere in the template prior to the `call-template` syntax:

```xml
<?import:xdo://APPCODE.TEMPLATE_CODE.lang.TERR?>
```

where

- `APPCODE` is the Application code you assigned to the subtemplate in the Template Manager. For example, if you associated this template with the Receivables application, enter "AR".
- `TEMPLATE_CODE` is the template Code you assigned to the subtemplate in the Template Manager. For example, AR_CommonComponents.
- `lang` is the two-letter ISO 639 language code for the template language you specified for the subtemplate in the Template Manager (for example, English is "en"). This entry must be lowercase.
- `TERR` is the two-letter ISO 3166 country code for the template territory you specified for the subtemplate in the Template Manager (for example, United States of America is "US"). This entry must be uppercase.

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

**Add the call-template Syntax to Your Primary Template**

Use the `call-template` command to render the contents of the subtemplate in the position specified, as follows:

```xml
<?call-template:template_name?>
```

where

- `template_name` is the name you assigned to the subtemplate in Step 1.

**Example**

In this example, your company address is a fixed string that appears in all your templates. Rather than reproduce the string in all your templates, you can place it in one subtemplate and reference it from all the others.

The common components subtemplate will contain the following:

```xml
<?template:MyAddress?>
R&G Corporation
500 Marine Parkway
Redwood Shores, CA 94065
<?end template?>
```
This defines the string in a subtemplate called MyAddress. Save this template as RTF and add it to the Template Manager using the following criteria:

- Application Code: CUSTOM
- Template Code: COMMONCOMPONENTS
- Language: English
- Territory: United States of America
- Select "Yes" for Subtemplate

Now this subtemplate and any functions therein can be referenced from any other template.

To call this subtemplate from another template, first import it by inserting the following syntax at the top of the calling template:

```xml
```

At the position in the template where you want to display the address, enter:

```xml
<?call-template:MyAddress?>
```

At runtime the contents of the MyAddress subtemplate will be fetched and rendered in the layout of the calling template.

This functionality is not limited to just strings, you can insert any valid RTF template functionality in a subtemplate, and even pass parameters from one to the other.

**Testing Your Subtemplate**

If have the XML Publisher Desktop Template Builder and Template Viewer installed, you can preview the template and subtemplate combination before uploading them to the Template Manager. To test from your local environment, you must alter the import template syntax to enable the XML Publisher processor to locate the subtemplate file.

To test, enter the import template syntax as follows:

```xml
<?import:file:path to subtemplate file?>
```

where `path to subtemplate file` is the location in your local environment where the subtemplate file is located. For example:

```xml
<?import:file:C:///Template_Directory/subtemplate_file.rtf?>
```

Now when you select the **Preview** option in the Template Builder, the XML Publisher processor can locate your subtemplate and render it from your local environment.

Note that before you upload the primary template to the Template Manager you must change the import syntax as specified in the step Enter the Import Syntax in the Primary Template, page 2-132.
Creating a PDF Template

This chapter covers the following topics:

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

PDF Template Overview

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

• Design the layout of your template using any application that generates documents that can be converted to PDF

• Scan a paper document to use as a template

• Download a PDF document from a third-party Web site

Note: The steps required to create a template from a third-party PDF depend on whether form fields have been added to the document. For more information, see Creating a Template from a Downloaded PDF, page 3-17.
If you are designing the layout, note that once you have converted to PDF, your layout is treated like a set background. When you mark up the template, you draw fields on top of this background. To edit the layout, you must edit your original document and then convert back to PDF.

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

**Supported Modes**

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

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

**Designing the Layout**

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

The following is the layout for a sample purchase order. It was designed using Microsoft Word and converted to PDF using Adobe Acrobat Distiller.
The following is the XML data that will be used as input to this template:
Adding Markup to the Template Layout

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

The process of associating the XML data to the PDF template is the same as the process for the RTF template. See: Associating the XML data to the template layout, page 2-4.

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

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

**Note:** If you are using Adobe Acrobat 5.0, the Form Tool is available from the standard toolbar. If you are using Adobe Acrobat 6.0 or later, display the Forms Toolbar from the Tools menu by selecting Tools > Advanced Editing > Forms > Show Forms Toolbar.

Naming the Placeholder

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

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

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

For information on mapping fields in the Template Manager, see Mapping PDF Template Fields, *Oracle XML Publisher Administration and Developer’s Guide*.

Creating a Text Placeholder

To create a text placeholder in your PDF document:

**Acrobat 5.0 Users:**

1. Select the Form Tool from the Acrobat toolbar.
2. Draw a form field box in the position on the template where you want the field to display. Drawing the field opens the Field Properties dialog box.
3. In the Name field of the Field Properties dialog box, enter a name for the field.
4. Select Text from the Type drop down menu.
   
   You can use the Field Properties dialog box to set other attributes for the placeholder. For example, enforce maximum character size, set field data type, data type validation, visibility, and formatting.
5. If the field is not placed exactly where desired, drag the field for exact placement.
Acrobat 6.0 (and later) Users:

1. Select the Text Field Tool from the Forms Toolbar.

2. Draw a form field box in the position on the template where you want the field to display. Drawing the field opens the Text Field Properties dialog box.

3. On the General tab, enter a name for the placeholder in the Name field.
   
   You can use the Text Field Properties dialog box to set other attributes for the placeholder. For example, enforce maximum character size, set field data type, data type validation, visibility, and formatting.

4. If the field is not placed exactly where desired, drag the field for exact placement.

Supported Field Properties Options

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

- **General**
  - Read Only
    
    The setting of this check box in combination with a set of configuration properties control the read-only/updateable state of the field in the output PDF. See Setting Fields as Updateable or Read Only, page 3-16.

- **Appearance**
  - Border Settings: color, background, width, and style
  - Text Settings: color, font, size
  - Common Properties: read only, required, visible/hidden, orientation (in degrees)
    
    (In Acrobat 6.0, these are available from the General tab)
  - Border Style

- **Options** tab
  - Multi-line
  - Scrolling Text

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

**Creating a Check Box**

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

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

To create a check box field:

**Acrobat 5.0 Users:**

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

**Acrobat 6.0 (and later) Users:**

1. Select the Check Box Tool from the Forms Toolbar.
2. Draw the check box field in the desired position.
3. On the General tab of the Check Box Properties dialog box, enter a Name for the field.
4. Select the Options tab.
5. In the Export Value field enter the value that the XML data field should match to enable the "checked" state.
   
   For the example, enter "Y" for each check box field.

**Creating a Radio Button Group**

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

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

To create a radio button group:

**Acrobat 5.0 Users:**
1. Draw the form field.
2. On the Field Properties dialog box, enter a Name for the field. Each radio button you define to represent this value can be named differently, but must be mapped to the same XML data field.
3. Select Radio Button from the Type drop down list.
4. Select the Options tab.
5. In the Export Value field enter the value that the XML data field should match to enable the "on" state.
   
   For the example, enter "Standard" for the field labeled "Standard". Enter "Overnight" for the field labeled "Overnight".

**Acrobat 6.0 (and later) Users:**
1. Select the Radio Button Tool from the Forms Toolbar.
2. Draw the form field in the position desired on the template.
3. On the General tab of the Radio Button Properties dialog, enter a Name for the field. Each radio button you define to represent this value can be named differently, but must be mapped to the same XML data field.
4. Select the Options tab.
5. In the Export Value field enter the value that the XML data field should match to enable the "on" state.
   
   For the example, enter "Standard" for the field labeled "Standard". Enter "Overnight" for the field labeled "Overnight".

**Defining Groups of Repeating Fields**

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

1. Insert a form field at the beginning of the area that is to contain the group. (Acrobat 6.0 users select the Text Field Tool, then draw the form field.)

2. In the Name field of the Field Properties window, enter any unique name you choose. This field is not mapped.

3. Acrobat 5.0 users: Select Text from the Type drop down list.

4. In the Short Description field (Acrobat 5.0) or the Tooltip field (Acrobat 6.0) of the Field Properties window, enter the following syntax:
   ```xml
   <?rep_field="BODY_START"?>
   ```

5. Define the end of the group area by inserting a form field at the end of the area that is to contain the group.

6. In the Name field of the Field Properties window, enter any unique name you choose. This field is not mapped. Note that the name you assign to this field must be different from the name you assigned to the "body start" field.

7. Acrobat 5.0 users: Select Text from the Type drop down list.

8. In the Short Description field (Acrobat 5.0) or the Tooltip field (Acrobat 6.0) of the Field Properties window, enter the following syntax:
   ```xml
   <?rep_field="BODY_END"?>
   ```

To define a group of repeating fields:

1. Insert a placeholder for the first element of the group.
   
   **Note:** The placement of this field in relationship to the BODY_START tag defines the distance between the repeating rows for each occurrence. See Placement of Repeating Fields, page 3-15.

2. For each element in the group, enter the following syntax in the Short Description field (Acrobat 5.0) or the Tooltip field (Acrobat 6.0):
   ```xml
   <?rep_field="T1_Gn"?>
   ```
   where n is the row number of the item on the template.

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

   - For the fields belonging to the row that begins with "PO_LINE_NUM" enter
     ```xml
     <?rep_field="T1_G1"?>
     ```
   - For the fields belonging to the row that begins with "C_FLEX_ITEM_DISP"
enter

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

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

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

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

![Graphic showing field entries]

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

**Adding Page Numbers and Page Breaks**

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

- Page Numbers
- Page Breaks

**Adding Page Numbers**

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

1. Decide the position on the template where you want the page number to be displayed.

2. Create a placeholder field called `@pagenum@` (see Creating a Text Placeholder, page 3-5).

3. Enter a starting value for the page number in the **Default** field. If the XML data includes a value for this field, the start value assigned in the template will be overridden. If no start value is assigned, it will default to 1.

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

You can define a page break in your template to occur after a repeatable field. To insert a page break after the occurrence of a specific field, add the following to the syntax in the Short Description field of the Field Properties dialog box (use the Tooltip field for Acrobat 6.0):

```
page_break="yes"
```

For example:
```
<?rep_field="T1_G3", page_break="yes"?>
```

The following example demonstrates inserting a page break in a template. The XML sample contains salaries of employees by department:
<?xml version="1.0"?>
<! - Generated by Oracle Reports version 6.0.8.22.0 - >
<ROOT>
  <LIST_G_DEPTNO>
    <G_DEPTNO>10</G_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>30</G_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>9400</SUMSALPERDEPTNO>
  </G_DEPTNO>
  <LIST_G_DEPTNO>
    <G_DEPTNO>9150</G_DEPTNO>
  </LIST_G_DEPTNO>
  <LIST_G_DEPTNO>
    <G_DEPTNO>9400</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**

<table>
<thead>
<tr>
<th>Dept No.</th>
<th>Emp No</th>
<th>Emp Name</th>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SUMSALPERDEPTNO</td>
</tr>
</tbody>
</table>

To insert a page break after each department, insert the page break syntax in the Short Description (or Tooltip field) for the SUMSALPERDEPTNO field as follows:

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

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

![Field Properties Dialog Box](image)

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:
Performing Calculations

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

1. Create a text field to display the calculated total. Give the field any Name you choose.

2. In the Field Properties dialog box, select the Format tab.

3. Select Number from the Category list.

4. Select the Calculate tab.

5. Select the radio button next to "Value is the operation of the following fields:"

6. Select sum from the drop down list.

7. Select the Pick... button and select the fields that you want totaled.

Completed PDF Template

The following figure shows the completed PDF template:
Runtime Behavior

Placement of Repeating Fields

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

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

Second occurrence, single row:

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

Second occurrence, multiple rows:

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

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

Setting Fields as Updateable or Read Only

When you define a field in the template you have the option of selecting “Read Only” for the field, as shown in the following sample Text Field Properties dialog:

Regardless of what you choose at design time for the Read Only check box, the default behavior of the PDF processing engine is to set all fields to read-only for the output PDF. You can change this behavior using the following configuration properties in the XML Publisher Configuration File, Oracle XML Publisher Administration and Developer’s Guide:

• all-field-readonly
• all-fields-readonly-asis

• remove-pdf-fields

Note that in the first two options, you are setting a state for the field in the PDF output. The setting of individual fields can still be changed in the output using Adobe Acrobat Professional. Also note that because the fields are maintained, the data is still separate and can be extracted. In the third option, "remove-pdf-fields" the structure is flattened and no field/data separation is maintained.

To make all fields updateable:

Set the "all-field-readonly" property to "false". This sets the Read Only state to "false" for all fields regardless of the individual field settings at design time.

To make all fields read only:

This is the default behavior. No settings are required.

To maintain the Read Only check box selection for each field:

To maintain the setting of the Read Only check box on a field-by-field basis in the output PDF, set the property "all-fields-readonly-asis" to "true". This property will override the settings of "all-field-readonly".

To remove all fields from the output PDF:

Set the property "remove-pdf-fields" to "true".

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. Generate sample data from your Applications program that will be used as the data source for the PDF template.
   You must generate an XML schema for the data source in order to map the fields from the data source to the correct field positions on the template.

2. Register the Applications data source in the Template Manager (see Using the Template Manager, Oracle XML Publisher Administration and Developer’s Guide).

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

PDF forms downloaded from third party sources may or may not contain the form fields already defined. To determine if the form fields are defined, open the document in Adobe Acrobat and select the **Form Tool** (in Acrobat 6.0, select the **Text Field Tool**). If the form fields are defined, they will display in the document.

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

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

This chapter covers the following topics:

- Introduction
- Outbound eText Templates

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.

Outbound eText Templates

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

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

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

The graphic below illustrates the relationship between the XML data hierarchy and the
template Level. The XML element "RequestHeader" is defined as the Level. The data
elements defined in the table ("FileID" and "Encryption") are children of the
RequestHeader element.
The order of the tables in the template determines the print order of the records. At runtime the system loops through all the instances of the XML element corresponding to a table (Level) and prints the records belonging to the table. The system then moves on to the next table in the template. If tables are nested, the system will generate the nested records of the child tables before moving on to the next parent instance.

**Command Rows, Data Rows, and Data Column Header Rows**

The following figure shows the placement of Command Rows, Data Rows, and Data Column Header Rows:
Command rows are used to specify commands in the template. Command rows always have two columns: command name and command parameter. Command rows do not have column headings. The commands control the overall setup and record structures of the template.

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

Data Column Header Rows

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

The required data column header rows vary depending on the template type. See Structure of the Data Row, page 4-12.

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 4-12.
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

• FooterRecordB

Following would be its table layout:

<table>
<thead>
<tr>
<th>Level</th>
<th>Record</th>
</tr>
</thead>
<tbody>
<tr>
<td>RequestHeader</td>
<td></td>
</tr>
<tr>
<td>Batch</td>
<td></td>
</tr>
<tr>
<td>FileHeaderRecordA</td>
<td></td>
</tr>
<tr>
<td>BatchHeaderRecordA</td>
<td></td>
</tr>
<tr>
<td>Data rows for the BatchHeaderRecordA</td>
<td></td>
</tr>
<tr>
<td>BatchHeaderRecordB</td>
<td></td>
</tr>
<tr>
<td>Data rows for the BatchHeaderRecordB</td>
<td></td>
</tr>
<tr>
<td>Payment</td>
<td></td>
</tr>
<tr>
<td>PaymentRecordA</td>
<td></td>
</tr>
<tr>
<td>Data rows for the PaymentRecordA</td>
<td></td>
</tr>
<tr>
<td>PaymentRecordB</td>
<td></td>
</tr>
</tbody>
</table>
Multiple records for the same level can exist in the same table. However, each table can only have one level defined. In the example above, the BatchHeaderRecordA and BatchHeaderRecordB are both defined in the same table. However, note that the END
LEVEL for the Payment must be defined in its own separate table after the child element Invoice. The Payment END LEVEL cannot reside in the same table as the Invoice Level.

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

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

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

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

- Batch2
  - Payment1
    - Invoice1
    - Invoice2
    - Invoice3

This will generate the following printed records:

<table>
<thead>
<tr>
<th>Record Order</th>
<th>Record Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FileHeaderRecordA</td>
<td>One header record for the EFT file</td>
</tr>
<tr>
<td>Record Order</td>
<td>Record Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>2</td>
<td>BatchHeaderRecordA</td>
<td>For Batch1</td>
</tr>
<tr>
<td>3</td>
<td>BatchHeaderRecordB</td>
<td>For Batch1</td>
</tr>
<tr>
<td>4</td>
<td>PaymentRecordA</td>
<td>For Batch1, Payment1</td>
</tr>
<tr>
<td>5</td>
<td>PaymentRecordB</td>
<td>For Batch1, Payment1</td>
</tr>
<tr>
<td>6</td>
<td>InvoiceRecordA</td>
<td>For Batch1, Payment1, Invoice1</td>
</tr>
<tr>
<td>7</td>
<td>InvoiceRecordA</td>
<td>For Batch1, Payment1, Invoice2</td>
</tr>
<tr>
<td>8</td>
<td>PaymentRecordA</td>
<td>For Batch1, Payment2</td>
</tr>
<tr>
<td>9</td>
<td>PaymentRecordB</td>
<td>For Batch1, Payment2</td>
</tr>
<tr>
<td>10</td>
<td>InvoiceRecordA</td>
<td>For Batch1, Payment2, Invoice1</td>
</tr>
<tr>
<td>11</td>
<td>BatchFooterRecordC</td>
<td>For Batch1</td>
</tr>
<tr>
<td>12</td>
<td>BatchFooterRecordD</td>
<td>For Batch1</td>
</tr>
<tr>
<td>13</td>
<td>BatchHeaderRecordA</td>
<td>For Batch2</td>
</tr>
<tr>
<td>14</td>
<td>BatchHeaderRecordB</td>
<td>For Batch2</td>
</tr>
<tr>
<td>15</td>
<td>PaymentRecordA</td>
<td>For Batch2, Payment1</td>
</tr>
<tr>
<td>16</td>
<td>PaymentRecordB</td>
<td>For Batch2, Payment1</td>
</tr>
<tr>
<td>17</td>
<td>InvoiceRecordA</td>
<td>For Batch2, Payment2, Invoice1</td>
</tr>
<tr>
<td>18</td>
<td>InvoiceRecordA</td>
<td>For Batch2, Payment1, Invoice2</td>
</tr>
<tr>
<td>19</td>
<td>InvoiceRecordA</td>
<td>For Batch2, Payment1, Invoice3</td>
</tr>
</tbody>
</table>
New Record Command

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

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

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

The new record command can have a name as its parameter. This name becomes the name for the record. The record name is also referred to as the record type. The name can be used in the COUNT function for counting the generated instances of the record. See COUNT, page 4-28 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 Column**

Specifies the data type and format setting. There are three accepted data types:

- Alpha
- Number
- Date

Refer to Field Level Key Words, page 4-33 for their usage.

**Number Data Type**

Numeric data has three optional format settings: Integer, Decimal, or you can define a format mask. Specify the optional settings with the Number data type as follows:

- Number, Integer
- Number, Decimal
- Number, <format mask>

For example:

```
Number, ###,###.00
```

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 following table shows examples of how to set a format mask. When specifying the mask, # represents that a digit is to be displayed when present in the data; 0 represents that the digit placeholder is to be displayed whether data is present or not.

When specifying the format mask, the group separator must always be "," and the decimal separator must always be "." To alter these in the actual output, you must use the Setup Commands NUMBER THOUSANDS SEPARATOR and NUMBER DECIMAL SEPARATOR. See Setup Command Tables, page 4-16 for details on these commands.

The following table shows sample Data, Format Specifier, and Output. The Output assumes the default group and decimal separators.

<table>
<thead>
<tr>
<th>Data</th>
<th>Format Specifier</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456789</td>
<td>###,###.00</td>
<td>123,456,789.00</td>
</tr>
<tr>
<td>123456789.2</td>
<td>###.00</td>
<td>123456789.20</td>
</tr>
</tbody>
</table>
### Data Format Specifier Output

<table>
<thead>
<tr>
<th>Data</th>
<th>Format Specifier</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234.56789</td>
<td>###.000</td>
<td>1234.568</td>
</tr>
<tr>
<td>123456789.2</td>
<td>#</td>
<td>123456789</td>
</tr>
<tr>
<td>123456789.2</td>
<td>#.#</td>
<td>123456789.2</td>
</tr>
<tr>
<td>123456789</td>
<td>#.#</td>
<td>123456789</td>
</tr>
</tbody>
</table>

### Date Data Type

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

### Mapping EDI Delimiter-Based Data Types to eText Data Types

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

<table>
<thead>
<tr>
<th>ASC X12 Data Type</th>
<th>Format Template Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>A - Alphabetic</td>
<td>Alpha</td>
</tr>
<tr>
<td>AN - Alphanumeric</td>
<td>Alpha</td>
</tr>
<tr>
<td>B - Binary</td>
<td>Number</td>
</tr>
<tr>
<td>CD - Composite data element</td>
<td>N/A</td>
</tr>
<tr>
<td>CH - Character</td>
<td>Alpha</td>
</tr>
<tr>
<td>DT - Date</td>
<td>Date</td>
</tr>
<tr>
<td>FS - Fixed-length string</td>
<td>Alpha</td>
</tr>
<tr>
<td>ID - Identifier</td>
<td>Alpha</td>
</tr>
<tr>
<td>IV - Incrementing Value</td>
<td>Number</td>
</tr>
<tr>
<td>Nn - Numeric</td>
<td>Number</td>
</tr>
<tr>
<td>PW - Password</td>
<td>Alpha</td>
</tr>
</tbody>
</table>
Now assume you have specified the following setup commands:

| NUMBER THOUSANDS SEPARATOR | , |
| NUMBER DECIMAL SEPARATOR   | , |

The following table shows the Data, Format Specifier, and Output for this case. Note that the Format Specifier requires the use of the default separators, regardless of the setup command entries.

<table>
<thead>
<tr>
<th>Data</th>
<th>Format Specifier</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456789</td>
<td>###,###.00</td>
<td>123.456.789,00</td>
</tr>
<tr>
<td>123456789.2</td>
<td>###.00</td>
<td>123456789,20</td>
</tr>
<tr>
<td>1234.56789</td>
<td>###.000</td>
<td>1234,568</td>
</tr>
<tr>
<td>123456789.2</td>
<td>#</td>
<td>123456789</td>
</tr>
<tr>
<td>123456789.2</td>
<td>.##</td>
<td>123456789,2</td>
</tr>
<tr>
<td>123456789</td>
<td>.##</td>
<td>123456789</td>
</tr>
</tbody>
</table>

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

Example usage:

- To pad a field on the left with a "0", enter the following in the Pad column field:
  
  \text{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 4-27.

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
• Number Thousands Separator
• Number Decimal Separator
• Define Level
• Define Sequence
• Define Concatenation

Some example setup tables are shown in the following figures:
FORMAT SETUP:

Hint: Define formatting options...

<table>
<thead>
<tr>
<th>&lt;TEMPLATE TYPE&gt;</th>
<th>FIXED POSITION_BASED</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;OUTPUT CHARACTER SET&gt;</td>
<td>ISO-8859-1</td>
</tr>
<tr>
<td>&lt;NEW RECORD CHARACTER&gt;</td>
<td>Carriage Return</td>
</tr>
</tbody>
</table>

| <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.

<table>
<thead>
<tr>
<th>&lt;DEFINE LEVEL&gt;</th>
<th>PaymentsByPayDatePayee</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;BASE LEVEL&gt;</td>
<td>Payment</td>
</tr>
<tr>
<td>&lt;GROUPING CRITERIA&gt;</td>
<td>'PaymentDate, PayeeName'</td>
</tr>
<tr>
<td>&lt;END DEFINE LEVEL&gt;</td>
<td>PaymentsByPayDatePayee</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&lt;DEFINE LEVEL&gt;</th>
<th>InvoicesByReportingCatAndAttrib</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;BASE LEVEL&gt;</td>
<td>Invoice</td>
</tr>
</tbody>
</table>

Mapping of Payment Format:
International Payments EFT Format
This command specifies the type of template. There are two types: FIXED_POSITION_BASED and DELIMITER_BASED.

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

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

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

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

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

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

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

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

**Define Level Command**

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

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

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

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

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

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

- **Group Sort Ascending or Group Sort Descending** - defines the sorting of the group.
  Insert the <GROUP SORT ASCENDING> or <GROUP SORT DESCENDING>
command row anywhere between the `<DEFINE LEVEL>` and `<END DEFINE LEVEL>` commands. The parameter of the sort command is a comma-separated list of elements by which to sort the group.

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

<table>
<thead>
<tr>
<th>Payment Instance</th>
<th>PaymentDate (grouping criterion 1)</th>
<th>PayeeName (grouping criterion 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Payment1</td>
<td>PaymentDate1</td>
<td>PayeeName1</td>
</tr>
<tr>
<td>Payment2</td>
<td>PaymentDate2</td>
<td>PayeeName1</td>
</tr>
<tr>
<td>Payment3</td>
<td>PaymentDate1</td>
<td>PayeeName2</td>
</tr>
<tr>
<td>Payment4</td>
<td>PaymentDate1</td>
<td>PayeeName1</td>
</tr>
<tr>
<td>Payment5</td>
<td>PaymentDate1</td>
<td>PayeeName3</td>
</tr>
</tbody>
</table>

In the template, construct the setup table as follows to create a level called "PaymentsByPayDatePayee" from the base level "Payment" grouped according to PaymentDate and Payee Name. Add the Group Sort Ascending command to sort:

```
<DEFINE LEVEL> PaymentsByPayDatePayee
<Base LEVEL> Payment
<Grouping Criteria> PaymentDate, PayeeName
<Group Sort Ascending> PaymentDate, PayeeName
<End Define Level> PaymentsByPayDatePayee
```

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

<table>
<thead>
<tr>
<th>Payment Group Instance</th>
<th>Group Criteria</th>
<th>Payments in Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group1</td>
<td>PaymentDate1, PayeeName1</td>
<td>Payment1, Payment4</td>
</tr>
<tr>
<td>Group2</td>
<td>PaymentDate1, PayeeName2</td>
<td>Payment3</td>
</tr>
</tbody>
</table>
The order of the new instances is the order that the records will print. When evaluating the multiple grouping criteria to form the instances of the new level, the criteria can be thought of as forming a hierarchy. The first criterion is at the top of the hierarchy, the last criterion is at the bottom of the hierarchy.

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

**Example**

First Scenario: Group Records Only

EFT File Structure:

- BatchRec
  - PaymentGroupHeaderRec
  - PaymentGroupFooterRec

<table>
<thead>
<tr>
<th>Record Sequence</th>
<th>Record Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BatchRec</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PaymentGroupHeaderRec</td>
<td>For group 1 (PaymentDate1, PayeeName1)</td>
</tr>
<tr>
<td>3</td>
<td>PaymentGroupFooterRec</td>
<td>For group 1 (PaymentDate1, PayeeName1)</td>
</tr>
<tr>
<td>4</td>
<td>PaymentGroupHeaderRec</td>
<td>For group 2 (PaymentDate1, PayeeName2)</td>
</tr>
<tr>
<td>5</td>
<td>PaymentGroupFooterRec</td>
<td>For group 2 (PaymentDate1, PayeeName2)</td>
</tr>
<tr>
<td>6</td>
<td>PaymentGroupHeaderRec</td>
<td>For group 3 (PaymentDate1, PayeeName3)</td>
</tr>
<tr>
<td>7</td>
<td>PaymentGroupFooterRec</td>
<td>For group 3 (PaymentDate1, PayeeName3)</td>
</tr>
<tr>
<td>8</td>
<td>PaymentGroupHeaderRec</td>
<td>For group 4 (PaymentDate2, PayeeName1)</td>
</tr>
</tbody>
</table>
Example
Scenario 2: Group Records and Individual Records

EFT File Structure:
BatchRec
  • PaymentGroupHeaderRec
    • PaymentRec
  • PaymentGroupFooterRec

Generated output:

<table>
<thead>
<tr>
<th>Record Sequence</th>
<th>Record Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>PaymentGroupFooterRec</td>
<td>For group 4 (PaymentDate2, PayeeName1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Record Sequence</th>
<th>Record Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BatchRec</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PaymentGroupHeaderRec</td>
<td>For group 1 (PaymentDate1, PayeeName1)</td>
</tr>
<tr>
<td>3</td>
<td>PaymentRec</td>
<td>For Payment1</td>
</tr>
<tr>
<td>4</td>
<td>PaymentRec</td>
<td>For Payment4</td>
</tr>
<tr>
<td>5</td>
<td>PaymentGroupFooterRec</td>
<td>For group 1 (PaymentDate1, PayeeName1)</td>
</tr>
<tr>
<td>6</td>
<td>PaymentGroupHeaderRec</td>
<td>For group 2 (PaymentDate1, PayeeName2)</td>
</tr>
<tr>
<td>7</td>
<td>PaymentRec</td>
<td>For Payment3</td>
</tr>
<tr>
<td>8</td>
<td>PaymentGroupFooterRec</td>
<td>For group 2 (PaymentDate1, PayeeName2)</td>
</tr>
<tr>
<td>9</td>
<td>PaymentGroupHeaderRec</td>
<td>For group 3 (PaymentDate1, PayeeName3)</td>
</tr>
<tr>
<td>10</td>
<td>PaymentRec</td>
<td>For Payment5</td>
</tr>
<tr>
<td>11</td>
<td>PaymentGroupFooterRec</td>
<td>For group 3 (PaymentDate1, PayeeName3)</td>
</tr>
</tbody>
</table>
Once defined with the Define Level command, the new level can be used in the
template in the same manner as a level occurring in the extract. However, the records of
the new level can only reference the base level fields that are defined in its grouping
criteria. They cannot reference other base level fields other than in summary functions.

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

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

Define Sequence Command

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

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

Reset at Level

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

In some cases the sequence is reset outside the template. For example, a periodic
sequence may be defined to reset by date. In these cases, the PERIODIC_SEQUENCE
keyword is used for the reset at level. The system saves the last sequence number used
for a payment file to the database. Outside events control resetting the sequence in the
database. For the next payment file run, the sequence number is extracted from the
database for the start at number (see start at subcommand).
Increment Basis
The increment basis subcommand specifies if the sequence should be incremented based on record or extract instances. The allowed parameters for this subcommand are RECORD and LEVEL.

Enter RECORD to increment the sequence for every record.

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

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

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

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

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

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

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

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

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

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

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

```
<DEFINE CONCATENATION> ConcatenatedInvoiceInfo

<Base level> Invoice
<element> InvoiceNum
<delimiter> ','
</DEFINE CONCATENATION>
```

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

```
car_parts_inv0001, car_parts_inv0002, car_parts_inv0003, car_parts_inv0004, car_parts_inv0005
```

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

```
TRUNCATE(ConcatenatedInvoiceInfo, 40)
SUBSTR(ConcatenatedInvoiceInfo, 1, 40)
```

Either of these statements will result in:

```
car_parts_inv0001, car_parts_inv0002, car
```

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

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

to get the following string:

```
parts_inv0003, car_parts_inv0004, car_par
```
Invalid Characters and Replacement Characters Commands

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

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

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

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

First name?+Last name equates to 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.

**Number Thousands Separator and Number Decimal Separator**

The default thousands (or group) separator is a comma (“,”) and the default decimal separator is “.". Use the Number Thousands Separator command and the Number Decimal Separator command to specify separators other than the defaults. For example, to define “.” as the group separator and “,” as the decimal separator, enter the following:

```
NUMBER THOUSANDS SEPARATOR .

NUMBER DECIMAL SEPARATOR ,
```

For more information on formatting numbers, see Format Column, page 4-13.

**Expressions, Control Structures, 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:

\[\text{IF} \ <\text{boolean}\_\text{expressionA}> \ \text{THEN} \ \ <\text{numeric}\ or\ \text{character}\_\text{expression1}> \ \ [\text{ELSIF} \ <\text{boolean}\_\text{expressionB}\ \text{THEN} \ \ <\text{numeric}\ or\ \text{character}\_\text{expression2}>] \ ... \ \ [\text{ELSE} \ \ <\text{numeric}\ or\ \text{character}\_\text{expression3}>] \ \ \text{END IF}\]

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.

The IN predicate is supported in the IF-THEN-ELSE control structure. For example:

\[\text{IF} \ \text{PaymentAmount/Currency/Code} \ \text{IN} \ \{\text{`USD'}, \text{`EUR'}, \text{`AON'}, \text{`AZM'}\} \ \text{THEN} \ \]
\[\ \ \ \text{PayeeAccount/FundsCaptureOrder/OrderAmount/Value} \ * \ 100 \ \]
\[\text{ELSIF} \ \text{PaymentAmount/Currency/Code} \ \text{IN} \ \{\text{`BHD'}, \text{`IQD'}, \text{`KWD'}\} \ \text{THEN} \ \]
\[\ \ \ \text{PayeeAccount/FundsCaptureOrder/OrderAmount/Value} \ * \ 1000 \ \]
\[\text{ELSE} \ \]
\[\ \ \ \text{PayeeAccount/FundsCaptureOrder/OrderAmount/Value} \ \]
\[\ \ \ \text{END IF};\]

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) record belonging to the current (parent level) 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.

- **FORMAT_DATE** - Formats a date string to any desirable date format. For example:
  
  
  ```
  FORMAT_DATE("1900-01-01T18:19:20", "YYYY/MM/DD HH24:MI:SS")
  ```

  will produce the following output:

  ```
  1900/01/01 18:19:20
  ```

- **FORMAT_NUMBER** – Formats a number to display in desired format. For example:
  
  ```
  FORMAT_NUMBER("1234567890.0987654321", "999,999.99")
  ```

  produces the following output:

  ```
  1,234,567,890.10
  ```

- **MESSAGE_LENGTH** - returns the length of the message in the EFT message.

- **RECORD_LENGTH** - returns the length of the record in the EFT message.

- **INSTR** – returns the numeric position of a named character within a text field.

- **SYSDATE, DATE** – gets Current Date and Time.

- **POSITION** – returns the position of a node in the XML document tree structure.

- **REPLACE** – replaces a string with another string.

- **CONVERT_CASE** – converts a string or a character to UPPER or LOWER case.

- **CHR** – gets the character representation of an argument, which is an ASCII value.
• LPAD, RPAD – generates left or right padding for string values.

• AND, OR, NOT – operator functions on elements.

• Other SQL functions include the following. Use the syntax corresponding to the SQL function.
  • TO_DATE
  • LOWER
  • UPPER
  • LENGTH
  • GREATEST
  • LEAST
  • DECODE
  • CEIL
  • ABS
  • FLOOR
  • ROUND
  • CHR
  • TO_CHAR
  • SUBSTR
  • LTRIM
  • RTRIM
  • TRIM
  • IN
  • TRANSLATE

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 XML syntax.
**Symbol** | **Usage**  
---|---  
IN | Equal-to-any-member-of test.  
NOT IN | Negates the IN operator.  
| Not-Equal-to-any-member-of test.  

**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.
This chapter covers the following topics:

- Extended SQL and XSL Functions
- XSL Equivalents
- Using FO Elements

Extended SQL and XSL Functions

XML Publisher has extended a set of SQL and XSL functions for use in RTF templates. The syntax for these extended functions is

```xml
<?xdofx:expression?>
```

for extended SQL functions or

```xml
<?xdoxslt:expression?>
```

for extended XSL functions.

**Note:** You cannot mix `xdofx` statements with XSL expressions in the same context. For example, assume you had two elements, `FIRST_NAME` and `LAST_NAME` that you wanted to concatenate into a 30-character field and right pad the field with the character "x", you could NOT use the following:

**INCORRECT:**

```xml
<?xdofx:rpad(concat(FIRST_NAME,LAST_NAME),30,'x') ?>
```

because `concat` is an XSL expression. Instead, you could use the following:

**CORRECT:**

```xml
<?xdofx:rpad(FIRST_NAME||LAST_NAME,30,'x') ?>
```

The supported functions are shown in the following table:
<table>
<thead>
<tr>
<th>SQL Statement or XSL Expression</th>
<th>Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2+3</td>
<td>&lt;?xdofx:2+3?&gt;</td>
<td>Addition</td>
</tr>
<tr>
<td>2-3</td>
<td>&lt;?xdofx:2-3?&gt;</td>
<td>Subtraction</td>
</tr>
<tr>
<td>2*3</td>
<td>&lt;?xdofx:2*3?&gt;</td>
<td>Multiplication</td>
</tr>
<tr>
<td>2/3</td>
<td>&lt;?xdofx:2/3?&gt;</td>
<td>Division</td>
</tr>
<tr>
<td>2**3</td>
<td>&lt;?xdofx:2**3?&gt;</td>
<td>Exponential</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>lpad('aaa',10,'.')</td>
<td>&lt;?xdofx:lpad('aaa',10,'.')?&gt;</td>
<td>The lpad function pads the left side of a string with a specific set of characters. The syntax for the lpad function is: lpad(string1,padded_length,[pad_string]) string1 is the string to pad characters to (the left-hand side). padded_length is the number of characters to return. pad_string is the string that will be padded to the left-hand side of string1.</td>
</tr>
<tr>
<td>rpad('aaa',10,'.')</td>
<td>&lt;?xdofx:rpad('aaa',10,'.')?&gt;</td>
<td>The rpad function pads the right side of a string with a specific set of characters. The syntax for the rpad function is: rpad(string1,padded_length,[pad_string]). string1 is the string to pad characters to (the right-hand side). padded_length is the number of characters to return. pad_string is the string that will be padded to the right-hand side of string1.</td>
</tr>
<tr>
<td>SQL Statement or XSL Expression</td>
<td>Usage</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| decode('xxx','bbb','ccc','xxx', 'ddd') | `<xdo felony:decode('xxx','bbb','ccc','xxx', 'ddd')?>` | The decode function has the functionality of an IF-THEN-ELSE statement. The syntax for the decode function is: `decode(expression, search, result [,search, result]...[, default])`  
expression is the value to compare.  
search is the value that is compared against expression.  
result is the value returned, if expression is equal to search.  
default is returned if no matches are found. |
| Instr('abcabcabc','a',2) | `<xdo felony: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) | `<xdo felony: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. |
### SQL Statement or XSL Expression | Usage | Description |
---|---|---|
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:

\[
\text{replace(string1,string_to_replace,[replacement_string])}
\]

*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')>` | Function to_number converts char, a value of CHAR, VARCHAR2, NCHAR, or NVARCHAR2 datatype containing a number in the format specified by the optional format model *fmt*, to a value of NUMBER datatype.

to_char(12345) | `<xdofx:to_char('12345')>` | Use the TO_CHAR function to translate a value of NUMBER datatype to VARCHAR2 datatype.

to_date | `<xdofx:to_date ( char [,'fmt' [, 'nlsparam']])` | TO_DATE converts char of CHAR, VARCHAR2, NCHAR, or NVARCHAR2 datatype to a value of DATE datatype. The *fmt* is a date format specifying the format of *char*. If you omit *fmt*, then *char* must be in the default date format. If *fmt* is 'J', for Julian, then *char* must be an integer.

sysdate() | `<xdofx:sysdate()>` | SYSDATE returns the current date and time. The datatype of the returned value is DATE. The function requires no arguments.

minimum | `<xdofxslt:minimum(ELEMENT_NAME)>` | Returns the minimum value of the element in the set.

maximum | `<xdofxslt:maximum(ELEMENT_NAME)>` | Returns the maximum value of the element in the set.
<table>
<thead>
<tr>
<th>SQL Statement or XSL Expression</th>
<th>Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chr</td>
<td><code>&lt;xdofx:chr(n)?&gt;</code></td>
<td>CHR returns the character having the binary equivalent to <code>n</code> in either the database character set or the national character set.</td>
</tr>
<tr>
<td>ceil</td>
<td><code>&lt;xdofx:ceil(n)?&gt;</code></td>
<td>CEIL returns smallest integer greater than or equal to <code>n</code>.</td>
</tr>
<tr>
<td>floor</td>
<td><code>&lt;xdofx:floor(n)?&gt;</code></td>
<td>FLOOR returns largest integer equal to or less than <code>n</code>.</td>
</tr>
<tr>
<td>round</td>
<td><code>&lt;xdofx:round ( number [, integer ] )?&gt;</code></td>
<td>ROUND returns <code>number</code> rounded to <code>integer</code> places right of the decimal point. If <code>integer</code> is omitted, then <code>number</code> is rounded to 0 places. <code>integer</code> can be negative to round off digits left of the decimal point. <code>integer</code> must be an integer.</td>
</tr>
<tr>
<td>lower</td>
<td><code>&lt;xdofx:lower (char)?&gt;</code></td>
<td>LOWER returns <code>char</code>, with all letters lowercase. <code>char</code> can be any of the datatypes CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB. The return value is the same datatype as <code>char</code>.</td>
</tr>
<tr>
<td>upper</td>
<td><code>&lt;xdofx:upper(char)?&gt;</code></td>
<td>UPPER returns <code>char</code>, with all letters uppercase. <code>char</code> can be any of the datatypes CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB. The return value is the same datatype as <code>char</code>.</td>
</tr>
<tr>
<td>length</td>
<td><code>&lt;xdofx:length(char)?&gt;</code></td>
<td>The &quot;length&quot; function returns the length of <code>char</code>. LENGTH calculates length using characters as defined by the input character set.</td>
</tr>
<tr>
<td>greatest</td>
<td><code>&lt;xdofx:greatest ( expr [, expr]... )?&gt;</code></td>
<td>GREATEST returns the greatest of the list of <code>exprs</code>. All <code>exprs</code> after the first are implicitly converted to the datatype of the first <code>expr</code> before the comparison.</td>
</tr>
<tr>
<td>least</td>
<td><code>&lt;xdofx:least ( expr [, expr]... )?&gt;</code></td>
<td>LEAST returns the least of the list of <code>exprs</code>. All <code>exprs</code> after the first are implicitly converted to the datatype of the first <code>expr</code> before the comparison.</td>
</tr>
</tbody>
</table>
The following table shows supported combination functions:

<table>
<thead>
<tr>
<th>SQL Statement</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2+3/4-6*7)/8</td>
<td>&lt;?xdofx:(2+3/4-6*7)/8?&gt;</td>
</tr>
<tr>
<td>lpad(substr('1234567890',5,3),10,'^')</td>
<td>&lt;?xdofx:lpad(substr('1234567890',5,3),10,'^')?&gt;</td>
</tr>
<tr>
<td>decode('a','b','c','d','e','1')</td>
<td></td>
</tr>
</tbody>
</table>

**XSL Equivalents**

The following table lists the XML Publisher simplified syntax with the XSL equivalents.

<table>
<thead>
<tr>
<th>Supported XSL Elements</th>
<th>Description</th>
<th>XML Publisher Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;xsl:value-of select=&quot;name&quot;&gt;</td>
<td>Placeholder syntax</td>
<td>&lt;?name?&gt;</td>
</tr>
<tr>
<td>&lt;xsl:apply-templates select=&quot;name&quot;&gt;</td>
<td>Applies a template rule to the current element's child nodes.</td>
<td>&lt;?apply:name?&gt;</td>
</tr>
<tr>
<td>&lt;xsl:copy-of select=&quot;name&quot;&gt;</td>
<td>Creates a copy of the current node.</td>
<td>&lt;?copy-of:name?&gt;</td>
</tr>
<tr>
<td>&lt;xsl:call-template name=&quot;name&quot;&gt;</td>
<td>Calls a named template to be inserted into/applied to the current template.</td>
<td>&lt;?call:name?&gt;</td>
</tr>
<tr>
<td>&lt;xsl:sort select=&quot;name&quot;&gt;</td>
<td>Sorts a group of data based on an element in the dataset.</td>
<td>&lt;?sort:name?&gt;</td>
</tr>
<tr>
<td>&lt;xsl:for-each select=&quot;name&quot;&gt;</td>
<td>Loops through the rows of data of a group, used to generate tabular output.</td>
<td>&lt;?for-each:name?&gt;</td>
</tr>
<tr>
<td><a href="">xsl:choose</a></td>
<td>Used in conjunction with when and otherwise to express multiple conditional tests.</td>
<td>&lt;?choose?&gt;</td>
</tr>
</tbody>
</table>
### Supported XSL Elements

<table>
<thead>
<tr>
<th>Supported XSL Elements</th>
<th>Description</th>
<th>XML Publisher Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;xsl:when test=&quot;exp&quot;&gt;</code></td>
<td>Used in conjunction with <code>choose</code> and <code>otherwise</code> to express multiple conditional tests</td>
<td><code>&lt;?when:expression?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xsl:otherwise&gt;</code></td>
<td>Used in conjunction with <code>choose</code> and <code>when</code> to express multiple conditional tests</td>
<td><code>&lt;?otherwise?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xsl:if test=&quot;exp&quot;&gt;</code></td>
<td>Used for conditional formatting.</td>
<td><code>&lt;?if:expression?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xsl:template name=&quot;name&quot;&gt;</code></td>
<td>Template declaration</td>
<td><code>&lt;?template:name?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xsl:variable name=&quot;name&quot;&gt;</code></td>
<td>Local or global variable declaration</td>
<td><code>&lt;?variable:name?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xsl:import href=&quot;url&quot;&gt;</code></td>
<td>Import the contents of one stylesheet into another</td>
<td><code>&lt;?import:url?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xsl:include href=&quot;url&quot;&gt;</code></td>
<td>Include one stylesheet in another</td>
<td><code>&lt;?include:url?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xsl:stylesheet xmlns:x=&quot;url&quot;&gt;</code></td>
<td>Define the root element of a stylesheet</td>
<td><code>&lt;?namespace:x=url?&gt;</code></td>
</tr>
</tbody>
</table>

### Using FO Elements

You can use most FO elements in an RTF template inside the Microsoft Word form fields. The following FO elements have been extended for use with XML Publisher RTF templates. The XML Publisher syntax can be used with either RTF template method.

The full list of FO elements supported by XML Publisher can be found in the Appendix: Supported XSL-FO Elements, page A-1.
Supported XSL-FO Elements

The following table lists the XSL-FO elements supported in this release of BI Publisher. For each element the supported content elements and attributes are listed. If elements have shared supported attributes, these are noted as a group and are listed in the subsequent table, Property Groups. For example, several elements share the content element `inline`. Rather than list the `inline` properties each time, each entry notes that "inline-properties" are supported. The list of inline-properties can then be found in the Property Groups table.

<table>
<thead>
<tr>
<th>Element</th>
<th>Supported Content Elements</th>
<th>Supported Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic-link</td>
<td>external-graphic</td>
<td>inline-properties</td>
</tr>
<tr>
<td></td>
<td>inline</td>
<td>external-destination</td>
</tr>
<tr>
<td></td>
<td>leader</td>
<td>internal-destination</td>
</tr>
<tr>
<td></td>
<td>page-number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>page-number-citation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>basic-link</td>
<td></td>
</tr>
<tr>
<td></td>
<td>block</td>
<td></td>
</tr>
<tr>
<td></td>
<td>block-container</td>
<td></td>
</tr>
<tr>
<td></td>
<td>table</td>
<td></td>
</tr>
<tr>
<td></td>
<td>list-block</td>
<td></td>
</tr>
<tr>
<td></td>
<td>wrapper</td>
<td></td>
</tr>
<tr>
<td>Element</td>
<td>Supported Content Elements</td>
<td>Supported Attributes</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>bidi-override</td>
<td>bidi-override</td>
<td>inline-properties</td>
</tr>
<tr>
<td></td>
<td>external-graphic</td>
<td></td>
</tr>
<tr>
<td></td>
<td>instream-foreign-object</td>
<td></td>
</tr>
<tr>
<td></td>
<td>inline</td>
<td></td>
</tr>
<tr>
<td></td>
<td>leader</td>
<td></td>
</tr>
<tr>
<td></td>
<td>page-number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>page-number-citation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>basic-link</td>
<td></td>
</tr>
<tr>
<td>block</td>
<td>external-graphic</td>
<td>block-properties</td>
</tr>
<tr>
<td></td>
<td>inline</td>
<td></td>
</tr>
<tr>
<td></td>
<td>page-number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>page-number-citation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>basic-link</td>
<td></td>
</tr>
<tr>
<td></td>
<td>block</td>
<td></td>
</tr>
<tr>
<td></td>
<td>block-container</td>
<td></td>
</tr>
<tr>
<td></td>
<td>table</td>
<td></td>
</tr>
<tr>
<td></td>
<td>list-block</td>
<td></td>
</tr>
<tr>
<td></td>
<td>wrapper</td>
<td></td>
</tr>
<tr>
<td>block-container</td>
<td>block</td>
<td>block-properties</td>
</tr>
<tr>
<td></td>
<td>block-container</td>
<td></td>
</tr>
<tr>
<td></td>
<td>table</td>
<td></td>
</tr>
<tr>
<td></td>
<td>list-block</td>
<td></td>
</tr>
<tr>
<td></td>
<td>wrapper</td>
<td></td>
</tr>
<tr>
<td>bookmark-tree</td>
<td>bookmark</td>
<td>N/A</td>
</tr>
<tr>
<td>Element</td>
<td>Supported Content Elements</td>
<td>Supported Attributes</td>
</tr>
<tr>
<td>----------------------------</td>
<td>----------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>bookmark</td>
<td>bookmark</td>
<td>external-destination</td>
</tr>
<tr>
<td></td>
<td>bookmark-title</td>
<td>internal-destination</td>
</tr>
<tr>
<td></td>
<td></td>
<td>starting-state</td>
</tr>
<tr>
<td>bookmark-title</td>
<td>N/A</td>
<td>color</td>
</tr>
<tr>
<td></td>
<td></td>
<td>font-style</td>
</tr>
<tr>
<td></td>
<td></td>
<td>font-weight</td>
</tr>
<tr>
<td>conditional-page-master-reference</td>
<td>N/A</td>
<td>master-reference</td>
</tr>
<tr>
<td></td>
<td></td>
<td>page-position</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• first</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• last</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• rest</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• any</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• inherit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>odd-or-even</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• odd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• even</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• any</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• inherit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>blank-or-not-blank</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• blank</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• not-blank</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• any</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• inherit</td>
</tr>
<tr>
<td>Element</td>
<td>Supported Content Elements</td>
<td>Supported Attributes</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>external-graphic</td>
<td>N/A</td>
<td>graphic-properties</td>
</tr>
<tr>
<td></td>
<td></td>
<td>src</td>
</tr>
<tr>
<td>flow</td>
<td>block</td>
<td>flow-properties</td>
</tr>
<tr>
<td></td>
<td>block-container</td>
<td></td>
</tr>
<tr>
<td></td>
<td>table</td>
<td></td>
</tr>
<tr>
<td></td>
<td>list-block</td>
<td></td>
</tr>
<tr>
<td></td>
<td>wrapper</td>
<td></td>
</tr>
<tr>
<td>inline</td>
<td>external-graphic</td>
<td>inline-properties</td>
</tr>
<tr>
<td></td>
<td>inline</td>
<td></td>
</tr>
<tr>
<td></td>
<td>leader</td>
<td></td>
</tr>
<tr>
<td></td>
<td>page-number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>page-number-citation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>basic-link</td>
<td></td>
</tr>
<tr>
<td></td>
<td>block</td>
<td></td>
</tr>
<tr>
<td></td>
<td>block-container</td>
<td></td>
</tr>
<tr>
<td></td>
<td>table</td>
<td></td>
</tr>
<tr>
<td></td>
<td>wrapper</td>
<td></td>
</tr>
<tr>
<td>instream-foreign-object</td>
<td>N/A</td>
<td>graphic-properties</td>
</tr>
<tr>
<td>layout-master-set</td>
<td>page-sequence-master</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>simple-page-master</td>
<td></td>
</tr>
<tr>
<td></td>
<td>simple-page-master</td>
<td></td>
</tr>
<tr>
<td></td>
<td>page-sequence-master</td>
<td></td>
</tr>
<tr>
<td>leader</td>
<td>N/A</td>
<td>inline-properties</td>
</tr>
<tr>
<td>list-block</td>
<td>list-item</td>
<td>block-properties</td>
</tr>
<tr>
<td>Element</td>
<td>Supported Content Elements</td>
<td>Supported Attributes</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>list-item</td>
<td>list-item-label</td>
<td>block-properties</td>
</tr>
<tr>
<td></td>
<td>list-item-body</td>
<td>block-properties</td>
</tr>
<tr>
<td>list-item-body</td>
<td>block</td>
<td>block-properties</td>
</tr>
<tr>
<td></td>
<td>block-container</td>
<td></td>
</tr>
<tr>
<td></td>
<td>table</td>
<td></td>
</tr>
<tr>
<td></td>
<td>list-block</td>
<td></td>
</tr>
<tr>
<td></td>
<td>wrapper</td>
<td></td>
</tr>
<tr>
<td>list-item-label</td>
<td>block</td>
<td>block-properties</td>
</tr>
<tr>
<td></td>
<td>block-container</td>
<td></td>
</tr>
<tr>
<td></td>
<td>table</td>
<td></td>
</tr>
<tr>
<td></td>
<td>list-block</td>
<td></td>
</tr>
<tr>
<td></td>
<td>wrapper</td>
<td></td>
</tr>
<tr>
<td>page-number</td>
<td>N/A</td>
<td>empty-inline-properties</td>
</tr>
<tr>
<td>page-number-citation</td>
<td>N/A</td>
<td>ref-id</td>
</tr>
<tr>
<td>Element</td>
<td>Supported Content Elements</td>
<td>Supported Attributes</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>page-sequence</td>
<td>static-content</td>
<td>inheritable-properties</td>
</tr>
<tr>
<td></td>
<td>flow</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>master-reference</td>
</tr>
<tr>
<td></td>
<td></td>
<td>initial-page-number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>force-page-count</td>
</tr>
<tr>
<td></td>
<td>• auto</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• end-on-even</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• end-on-odd</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• end-on-even-layout</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• end-on-odd-layout</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• no-force</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• inherit</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>format</td>
</tr>
<tr>
<td>page-sequence-master</td>
<td>single-page-master-reference</td>
<td>master-name</td>
</tr>
<tr>
<td></td>
<td>repeatable-page-master-reference</td>
<td></td>
</tr>
<tr>
<td></td>
<td>repeatable-page-master-alternatives</td>
<td></td>
</tr>
<tr>
<td>region-after</td>
<td>N/A</td>
<td>side-region-properties</td>
</tr>
<tr>
<td>region-before</td>
<td>N/A</td>
<td>side-region-properties</td>
</tr>
<tr>
<td>region-body</td>
<td>N/A</td>
<td>region-properties</td>
</tr>
<tr>
<td></td>
<td></td>
<td>margin-properties-CSS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>column-count</td>
</tr>
<tr>
<td>region-end</td>
<td>N/A</td>
<td>side-region-properties</td>
</tr>
<tr>
<td>Element</td>
<td>Supported Content Elements</td>
<td>Supported Attributes</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>region-start</td>
<td>N/A</td>
<td>side-region-properties</td>
</tr>
<tr>
<td>repeatable-page-master-alternatives</td>
<td>conditional-page-master-reference</td>
<td>maximum-repeats</td>
</tr>
<tr>
<td>repeatable-page-master-reference</td>
<td>N/A</td>
<td>master-reference</td>
</tr>
<tr>
<td>root</td>
<td>bookmark-tree</td>
<td>inheritable-properties</td>
</tr>
<tr>
<td></td>
<td>layout-master-set</td>
<td></td>
</tr>
<tr>
<td></td>
<td>page-sequence</td>
<td></td>
</tr>
<tr>
<td>Element</td>
<td>Supported Content Elements</td>
<td>Supported Attributes</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>simple-page-master</td>
<td>region-body</td>
<td>margin-properties-CSS</td>
</tr>
<tr>
<td></td>
<td>region-before</td>
<td>master-name</td>
</tr>
<tr>
<td></td>
<td>region-after</td>
<td>page-height</td>
</tr>
<tr>
<td></td>
<td>region-start</td>
<td>page-width</td>
</tr>
<tr>
<td></td>
<td>region-end</td>
<td>reference-orientation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 90</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 180</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 270</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• -90</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• -180</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• -270</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0deg</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 90deg</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 180deg</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 270deg</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• -90deg</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• -180deg</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• -270deg</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• inherit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• lr-tb</td>
</tr>
</tbody>
</table>

writing-mode
<table>
<thead>
<tr>
<th>Element</th>
<th>Supported Content Elements</th>
<th>Supported Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>single-page-master-reference</td>
<td>N/A</td>
<td>master-reference</td>
</tr>
<tr>
<td>static-content</td>
<td>block</td>
<td>flow-properties</td>
</tr>
<tr>
<td></td>
<td>block-container</td>
<td></td>
</tr>
<tr>
<td></td>
<td>table</td>
<td></td>
</tr>
<tr>
<td></td>
<td>wrapper</td>
<td></td>
</tr>
<tr>
<td>table</td>
<td>table-column</td>
<td>block-properties</td>
</tr>
<tr>
<td></td>
<td>table-header</td>
<td></td>
</tr>
<tr>
<td></td>
<td>table-footer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>table-body</td>
<td></td>
</tr>
<tr>
<td>table-body</td>
<td>table-row</td>
<td>inheritable-properties</td>
</tr>
<tr>
<td></td>
<td>id</td>
<td></td>
</tr>
<tr>
<td>table-cell</td>
<td>block</td>
<td>block-properties</td>
</tr>
<tr>
<td></td>
<td>block-container</td>
<td>number-columns-spanned</td>
</tr>
<tr>
<td></td>
<td>table</td>
<td>number-rows-spanned</td>
</tr>
<tr>
<td></td>
<td>list-block</td>
<td></td>
</tr>
<tr>
<td></td>
<td>wrapper</td>
<td></td>
</tr>
<tr>
<td>table-column</td>
<td>N/A</td>
<td>inheritable-properties</td>
</tr>
<tr>
<td></td>
<td>column-number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>column-width</td>
<td></td>
</tr>
<tr>
<td></td>
<td>number-columns-repeated</td>
<td></td>
</tr>
<tr>
<td>table-footer</td>
<td>table-row</td>
<td>inheritable-properties</td>
</tr>
<tr>
<td></td>
<td>id</td>
<td></td>
</tr>
<tr>
<td>table-header</td>
<td>table-row</td>
<td>inheritable-properties</td>
</tr>
<tr>
<td></td>
<td>id</td>
<td></td>
</tr>
<tr>
<td>Element</td>
<td>Supported Content Elements</td>
<td>Supported Attributes</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>table-row</td>
<td>table-cell</td>
<td>inheritable-properties</td>
</tr>
<tr>
<td></td>
<td></td>
<td>id</td>
</tr>
<tr>
<td>wrapper</td>
<td>inline</td>
<td>inheritable-properties</td>
</tr>
<tr>
<td></td>
<td>page-number</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td>page-number-citation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>basic-link</td>
<td></td>
</tr>
<tr>
<td></td>
<td>block</td>
<td></td>
</tr>
<tr>
<td></td>
<td>block-container</td>
<td></td>
</tr>
<tr>
<td></td>
<td>table</td>
<td></td>
</tr>
<tr>
<td></td>
<td>wrapper</td>
<td></td>
</tr>
</tbody>
</table>

### Property Groups Table

The following table lists the supported properties belonging to the attribute groups defined in the preceding table.
<table>
<thead>
<tr>
<th>Property Group</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>area-properties</td>
<td>overflow (visible, hidden)</td>
</tr>
<tr>
<td></td>
<td>reference-orientation</td>
</tr>
<tr>
<td></td>
<td>• 0</td>
</tr>
<tr>
<td></td>
<td>• 90</td>
</tr>
<tr>
<td></td>
<td>• 180</td>
</tr>
<tr>
<td></td>
<td>• 270</td>
</tr>
<tr>
<td></td>
<td>• -90</td>
</tr>
<tr>
<td></td>
<td>• -180</td>
</tr>
<tr>
<td></td>
<td>• -270</td>
</tr>
<tr>
<td></td>
<td>• 0deg</td>
</tr>
<tr>
<td></td>
<td>• 90deg</td>
</tr>
<tr>
<td></td>
<td>• 180deg</td>
</tr>
<tr>
<td></td>
<td>• 270deg</td>
</tr>
<tr>
<td></td>
<td>• -90deg</td>
</tr>
<tr>
<td></td>
<td>• -180deg</td>
</tr>
<tr>
<td></td>
<td>• -270deg</td>
</tr>
<tr>
<td></td>
<td>• inherit</td>
</tr>
<tr>
<td></td>
<td>writing-mode (lr-tb, rl-tb, lr, rl)</td>
</tr>
<tr>
<td></td>
<td>baseline-shift (baseline, sub, super)</td>
</tr>
<tr>
<td></td>
<td>vertical-align</td>
</tr>
<tr>
<td>block-properties</td>
<td>inheritable-properties</td>
</tr>
<tr>
<td></td>
<td>id</td>
</tr>
<tr>
<td>Property Group</td>
<td>Properties</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>border-padding-background-properties</td>
<td>background-color</td>
</tr>
<tr>
<td></td>
<td>background-image</td>
</tr>
<tr>
<td></td>
<td>background-position-vertical</td>
</tr>
<tr>
<td></td>
<td>background-position-horizontal</td>
</tr>
<tr>
<td></td>
<td>border</td>
</tr>
<tr>
<td></td>
<td>border-after-color</td>
</tr>
<tr>
<td></td>
<td>border-after-style (none, dotted, dashed, solid, double)</td>
</tr>
<tr>
<td></td>
<td>border-after-width</td>
</tr>
<tr>
<td></td>
<td>border-before-color</td>
</tr>
<tr>
<td></td>
<td>border-before-style (none, solid)</td>
</tr>
<tr>
<td></td>
<td>border-before-width</td>
</tr>
<tr>
<td></td>
<td>border-bottom</td>
</tr>
<tr>
<td></td>
<td>border-bottom-color</td>
</tr>
<tr>
<td></td>
<td>border-bottom-style (none, dotted, dashed, solid, double)</td>
</tr>
<tr>
<td></td>
<td>border-bottom-width</td>
</tr>
<tr>
<td></td>
<td>border-color</td>
</tr>
<tr>
<td></td>
<td>border-end-color</td>
</tr>
<tr>
<td></td>
<td>border-end-style (none, dotted, dashed, solid, double)</td>
</tr>
<tr>
<td></td>
<td>border-end-width</td>
</tr>
<tr>
<td></td>
<td>border-left</td>
</tr>
<tr>
<td></td>
<td>border-left-color</td>
</tr>
<tr>
<td></td>
<td>border-left-style (none, dotted, dashed, solid, double)</td>
</tr>
<tr>
<td></td>
<td>border-left-width</td>
</tr>
<tr>
<td></td>
<td>border-right</td>
</tr>
<tr>
<td></td>
<td>border-right-color</td>
</tr>
<tr>
<td></td>
<td>border-right-style (none, dotted, dashed, solid, double)</td>
</tr>
<tr>
<td></td>
<td>border-right-width</td>
</tr>
<tr>
<td></td>
<td>border-start-color</td>
</tr>
<tr>
<td>Property Group</td>
<td>Properties</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>border-start-style (none, dotted, dashed, solid, double)</td>
</tr>
<tr>
<td></td>
<td>border-start-width</td>
</tr>
<tr>
<td></td>
<td>border-top</td>
</tr>
<tr>
<td></td>
<td>border-top-color</td>
</tr>
<tr>
<td></td>
<td>border-top-style (none, dotted, dashed, solid, double)</td>
</tr>
<tr>
<td></td>
<td>border-top-width</td>
</tr>
<tr>
<td></td>
<td>border-width</td>
</tr>
<tr>
<td></td>
<td>padding</td>
</tr>
<tr>
<td></td>
<td>padding-after</td>
</tr>
<tr>
<td></td>
<td>padding-before</td>
</tr>
<tr>
<td></td>
<td>padding-bottom</td>
</tr>
<tr>
<td></td>
<td>padding-end</td>
</tr>
<tr>
<td></td>
<td>padding-left</td>
</tr>
<tr>
<td></td>
<td>padding-right</td>
</tr>
<tr>
<td></td>
<td>padding-start</td>
</tr>
<tr>
<td></td>
<td>padding-top</td>
</tr>
<tr>
<td>box-size-properties</td>
<td>height</td>
</tr>
<tr>
<td></td>
<td>width</td>
</tr>
<tr>
<td>character-properties</td>
<td>font-properties</td>
</tr>
<tr>
<td></td>
<td>text-decoration</td>
</tr>
<tr>
<td>empty-inline-properties</td>
<td>character-properties</td>
</tr>
<tr>
<td></td>
<td>border-padding-background-properties</td>
</tr>
<tr>
<td></td>
<td>id</td>
</tr>
<tr>
<td></td>
<td>color</td>
</tr>
<tr>
<td>Property Group</td>
<td>Properties</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>flow-properties</td>
<td>inheritable-properties</td>
</tr>
<tr>
<td></td>
<td>id</td>
</tr>
<tr>
<td></td>
<td>flow-name</td>
</tr>
<tr>
<td>font-properties</td>
<td>font-family</td>
</tr>
<tr>
<td></td>
<td>font-size</td>
</tr>
<tr>
<td></td>
<td>font-style (normal, italic, oblique)</td>
</tr>
<tr>
<td></td>
<td>font-weight (normal, bold)</td>
</tr>
<tr>
<td></td>
<td>table-omit-header-at-break (TRUE, FALSE, inherit)</td>
</tr>
<tr>
<td></td>
<td>table-omit-footer-at-break (TRUE, FALSE, inherit)</td>
</tr>
<tr>
<td>graphic-properties</td>
<td>border-padding-background-properties</td>
</tr>
<tr>
<td></td>
<td>margin-properties-inline</td>
</tr>
<tr>
<td></td>
<td>box-size-properties</td>
</tr>
<tr>
<td></td>
<td>font-properties</td>
</tr>
<tr>
<td></td>
<td>keeps-and-breaks-properties-atomic</td>
</tr>
<tr>
<td></td>
<td>id</td>
</tr>
<tr>
<td>Property Group</td>
<td>Properties</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>inheritable-properties</td>
<td>border-padding-background-properties</td>
</tr>
<tr>
<td></td>
<td>box-size-properties</td>
</tr>
<tr>
<td></td>
<td>margin-properties-inline</td>
</tr>
<tr>
<td></td>
<td>area-properties</td>
</tr>
<tr>
<td></td>
<td>character-properties</td>
</tr>
<tr>
<td></td>
<td>line-related-properties</td>
</tr>
<tr>
<td></td>
<td>leader-properties</td>
</tr>
<tr>
<td></td>
<td>keeps-and-breaks-properties-block</td>
</tr>
<tr>
<td></td>
<td>color</td>
</tr>
<tr>
<td></td>
<td>absolute-position</td>
</tr>
<tr>
<td></td>
<td>• auto</td>
</tr>
<tr>
<td></td>
<td>• absolute</td>
</tr>
<tr>
<td></td>
<td>• fixed</td>
</tr>
<tr>
<td></td>
<td>• inherit</td>
</tr>
<tr>
<td>inline-properties</td>
<td>inheritable-properties</td>
</tr>
<tr>
<td></td>
<td>id</td>
</tr>
<tr>
<td>keeps-and-breaks-properties-atomic</td>
<td>break-after (auto, column, page)</td>
</tr>
<tr>
<td></td>
<td>break-before (auto, column)</td>
</tr>
<tr>
<td></td>
<td>keep-with-next</td>
</tr>
<tr>
<td></td>
<td>keep-with-next.within-page</td>
</tr>
<tr>
<td>keeps-and-breaks-properties-block</td>
<td>keeps-and-breaks-properties-inline</td>
</tr>
<tr>
<td>Property Group</td>
<td>Properties</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>keeps-and-breaks-properties-inline</td>
<td>keeps-and-breaks-properties-atomic</td>
</tr>
<tr>
<td></td>
<td>keep-together</td>
</tr>
<tr>
<td></td>
<td>keep-together.within-line</td>
</tr>
<tr>
<td></td>
<td>keep-together.within-column</td>
</tr>
<tr>
<td></td>
<td>keep-together.within-page</td>
</tr>
<tr>
<td>leader-properties</td>
<td>leader-pattern (rule, dots)</td>
</tr>
<tr>
<td></td>
<td>leader-length</td>
</tr>
<tr>
<td></td>
<td>leader-length.optimum (dotted, dashed, solid, double)</td>
</tr>
<tr>
<td></td>
<td>rule-thickness</td>
</tr>
<tr>
<td>line-related-properties</td>
<td>text-align (start, center, end, justify, left, right, inherit)</td>
</tr>
<tr>
<td></td>
<td>text-align-last (start, center, end, justify, left, right, inherit)</td>
</tr>
<tr>
<td></td>
<td>text-indent</td>
</tr>
<tr>
<td></td>
<td>linefeed-treatment (ignore, preserve, treat-as-space, treat-as-zero-width-space, inherit )</td>
</tr>
<tr>
<td></td>
<td>white-space-treatment (ignore, preserve, ignore-if-before-linefeed, ignore-if-after-linefeed, ignore-if-surrounding-linefeed, inherit)</td>
</tr>
<tr>
<td></td>
<td>white-space-collapse (FALSE, TRUE, inherit)</td>
</tr>
<tr>
<td></td>
<td>wrap-option (no-wrap, wrap, inherit)</td>
</tr>
<tr>
<td></td>
<td>direction (ltr)</td>
</tr>
<tr>
<td>margin-properties-block</td>
<td>margin-properties-CSS</td>
</tr>
<tr>
<td></td>
<td>space-after</td>
</tr>
<tr>
<td></td>
<td>space-after.optimum</td>
</tr>
<tr>
<td></td>
<td>space-before</td>
</tr>
<tr>
<td></td>
<td>space-before.optimum</td>
</tr>
<tr>
<td></td>
<td>start-indent</td>
</tr>
<tr>
<td></td>
<td>end-indent</td>
</tr>
<tr>
<td>Property Group</td>
<td>Properties</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>margin-properties-CSS</td>
<td>margin</td>
</tr>
<tr>
<td></td>
<td>margin-bottom</td>
</tr>
<tr>
<td></td>
<td>margin-left</td>
</tr>
<tr>
<td></td>
<td>margin-right</td>
</tr>
<tr>
<td></td>
<td>margin-top</td>
</tr>
<tr>
<td>margin-properties-inline</td>
<td>margin-properties-block</td>
</tr>
<tr>
<td></td>
<td>space-start</td>
</tr>
<tr>
<td></td>
<td>space-start.optimum</td>
</tr>
<tr>
<td></td>
<td>space-end</td>
</tr>
<tr>
<td></td>
<td>space-end.optimum</td>
</tr>
<tr>
<td></td>
<td>position</td>
</tr>
<tr>
<td></td>
<td>• static</td>
</tr>
<tr>
<td></td>
<td>• relative</td>
</tr>
<tr>
<td></td>
<td>• absolute</td>
</tr>
<tr>
<td></td>
<td>• fixed</td>
</tr>
<tr>
<td></td>
<td>• inherit</td>
</tr>
<tr>
<td></td>
<td>top</td>
</tr>
<tr>
<td></td>
<td>left</td>
</tr>
<tr>
<td>region-properties</td>
<td>border-padding-background-properties</td>
</tr>
<tr>
<td></td>
<td>area-properties</td>
</tr>
<tr>
<td></td>
<td>region-name</td>
</tr>
<tr>
<td>side-region-properties</td>
<td>region-properties</td>
</tr>
<tr>
<td></td>
<td>extent</td>
</tr>
</tbody>
</table>
A

alignment
  RTF template, 2-43

B

background support
  RTF templates, 2-48
barcode formatting, 2-120
bidirectional language alignment
  RTF template, 2-43
body tags
  PDF template, 3-9
  RTF template, 2-16
bookmarks
  generating PDF bookmarks from an RTF template, 2-58
  inserting in RTF templates, 2-55
brought forward/carried forward page totals, 2-76

C

calculations in PDF template, 3-14
calendar profile option, 2-117
calendar specification, 2-117
cell highlighting
  conditional in RTF templates, 2-72
charts
  building in RTF templates, 2-20
check box placeholder
  creating in PDF template, 3-7
check box support

D

date fields in RTF templates, 2-46
date formatting
  RTF templates, 2-103
downloaded PDFs as templates, 3-17
drawing support, 2-31
drop-down form field support
  RTF templates, 2-60
dynamic data columns, 2-99
  example, 2-100
dynamic table of contents in RTF template, 2-58

e

end on even page, 2-55
etext data tables, 4-6
etext template command rows, 4-6
etext template setup command table, 4-16
even page
  force report to end on, 2-55
fixed-width columns
   RTF templates, 2-44

FO
   supported elements, A-1

FO elements
   using in RTF templates, 2-130, 5-7

fonts
   external, 2-118
   setting up, 2-118

footers
   RTF template, 2-16
   for-each-group XSL 2.0 standard, 2-83

formatting options in PDF templates, 3-6

form field method
   inserting placeholders, 2-9

form field properties options in PDF template, 3-6

form fields in the PDF template, 3-4

groups
   basic RTF method, 2-13
   defining in PDF template, 3-8
   defining in RTF template, 2-12
      syntax, 2-12
   defining in RTF templates, 2-6
   form field method, 2-14
   grouping scenarios in RTF template, 2-12
   in RTF templates, 2-6

headers and footers
   different first page, 2-17
   different odd and even pages, 2-17
   inserting placeholders, 2-16
   multiple, 2-16
   resetting within one output file, 2-95
   RTF template, 2-16

hidden text
   support in RTF templates, 2-43

horizontal table break, 2-100
hyperlinks
   bookmarks, 2-55

dynamic, 2-55
   inserting in RTF template, 2-55
   internal, 2-55
   static, 2-55

if statements, 2-64, 2-64
IF statements
   in free-form text, 2-64
   if-then-else statements, 2-65
images
   including in RTF template, 2-18
importing templates, 2-131
IN predicate
   If-Then-Else control structure
      e-text templates, 4-28

last page
   support for special content, 2-52

markup
   adding to the PDF template, 3-4
   adding to the RTF template, 2-8
   multicolumn page support, 2-46
   multiple headers and footers
      RTF template, 2-16

Namespace support in RTF template, 2-124

native page breaks and page numbering, 2-42

nulls
   how to test for in XML data, 2-82

number formatting
   RTF templates, 2-103

overflow data in PDF templates, 3-17

page breaks
   PDF templates, 3-10
   RTF template, 2-42, 2-50
page breaks and page numbering
native support, 2-42
page number
setting initial
   RTF templates, 2-51
page numbers
   PDF templates, 3-10
   restarting within one output file, 2-95
   RTF template, 2-43
page totals
   brought forward/carried forward, 2-76
   inserting in RTF template, 2-74
parameters
   defining in RTF template, 2-90
PDF templates
   adding markup, 3-4
   completed example, 3-14
   defining groups, 3-8
   definition of, 3-1
   downloading PDFs to use as templates, 3-17
   overflow data, 3-17
   page breaks, 3-10
   page numbering, 3-10
   placeholders
      check box, 3-7
      naming, 3-5
      naming to avoid mapping, 3-5
      radio button group, 3-7
      text, 3-5
      types of, 3-5
   placement of repeating fields at runtime, 3-15
   runtime behaviors, 3-15
   sample purchase order template, 3-2
   saving as Adobe Acrobat 5.0 compatible, 3-2
   sources for document templates, 3-2
   supported modes, 3-2
   when to use, 3-2
placeholders
   basic RTF method, 2-8, 2-8
   form field RTF method, 2-8, 2-9
   in PDF templates, 3-4
   in RTF templates, 2-6
      defining, 2-6, 2-8
   inserting in the header and footer of RTF
   template, 2-16
PDF templates
   check box, 3-7
naming, 3-5
   naming to avoid mapping, 3-5
radio button group, 3-7
text, 3-5
types of, 3-5
process overview, 1-3
   design time, 1-3
   runtime, 1-4
properties
   setting at template level, 2-92
R
   radio button group
      creating in PDF templates, 3-7
      regrouping, 2-83
repeating elements
   See groups
Rich Text Format (RTF)
   definition, 2-2
row breaking
   preventing in RTF templates, 2-44
row formatting
   conditional, 2-70
RTF placeholders
   syntax, 2-8
RTF template
   adding markup, 2-8
   applying design elements, 2-7
   definition, 2-2
   designing, 2-3
   groups, 2-6
   including images, 2-18
   native formatting features, 2-42
   placeholders, 2-6
   prerequisites, 2-2
   sample template design, 2-4
   supported modes, 2-4
      basic method, 2-2
      form field method, 2-2
      using XSL or XSL:FO, 2-2
RTF template design
   headers and footers, 2-16
RTF template placeholders, 2-8
running totals
   RTF templates, 2-80
S
sample RTF template
  completed markup, 2-12
section context command, 2-95
setting the initial page number
  RTF templates, 2-51
shape support, 2-31
sorting
  RTF template, 2-82
SQL functions
  using in RTF templates, 2-121
  XML Publisher syntax for, 5-1
SQL functions extended for XML Publisher, 5-1
subtemplates
  defining in your template, 2-131
syntax
  RTF template placeholder, 2-8

T
table features
  fixed-width columns, 2-44
  preventing rows breaking across pages
    RTF template, 2-44
  text truncation, 2-45
table features
  repeating table headers
    RTF template, 2-44
  RTF template, 2-43
table of contents support
  RTF template, 2-58
  dynamic TOC, 2-58
Tables
  best practices for formatting, 2-130
tables
  horizontal table break, 2-100
Template Builder, 2-3
Template Manager
  description, 1-2
text placeholder
  creating in PDF template, 3-5
text truncation in tables, 2-45
totals
  brought forward/carried forward, 2-76
  inserting page totals in RTF template, 2-74
  running

RTF templates, 2-80

U
updateable variables
  RTF templates, 2-89
user interfaces, 1-2

V
variables
  RTF templates, 2-89

W
watermarks
  RTF templates, 2-48

X
XML data file
  example, 2-5
XML file
  how to read, 2-5
XPath Support in RTF Templates, 2-121
XSL:FO elements
  using in RTF templates, 2-121
XSL elements
  apply a template rule, 2-127
  call template, 2-128
  copy the current node, 2-128
  define the root element of the stylesheet, 2-129
  import stylesheet, 2-128
  template declaration, 2-128
  using in RTF templates, 2-127
  variable declaration, 2-128
XML Publisher syntax for, 5-6