Contents

Preface

1 Introduction
   Overview of the Oracle Business Intelligence Publisher Report Designer’s Guide...........1-1

2 Getting Started
   Accessing Business Intelligence Publisher Enterprise....................................................2-1
   Setting Preferences.........................................................................................................2-2

3 Viewing and Scheduling Reports
   Navigating the Reports Repository..............................................................................3-1
   Viewing a Report Through the Report Viewer..............................................................3-2
   Using the Online Analyzer...........................................................................................3-6
   Using the BI Publisher Analyzer for Excel.................................................................3-11
   Scheduling a Report.....................................................................................................3-15
   Scheduling a Report To Be Burst................................................................................3-20
   Using Oracle BI Publisher in Oracle Enterprise Performance Management Workspace....3-20
   Using Oracle BI Publisher in Oracle Smart Space, Fusion Edition............................3-22
   Managing Your Scheduled Reports..............................................................................3-24
   Viewing Report History and Saved Output.................................................................3-25

4 Creating a New Report
   Process Overview..........................................................................................................4-1
   Create the Report Entry and Specify General Properties............................................4-2
   Using the Report Editor...............................................................................................4-3
   Defining the Data Model.............................................................................................4-6
<table>
<thead>
<tr>
<th>Feature</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native Support</td>
<td>7-15</td>
</tr>
<tr>
<td><strong>Inserting Images and Charts</strong></td>
<td>7-17</td>
</tr>
<tr>
<td>Images</td>
<td>7-17</td>
</tr>
<tr>
<td>Chart Support</td>
<td>7-18</td>
</tr>
<tr>
<td><strong>Drawing, Shape, and Clip Art Support</strong></td>
<td>7-29</td>
</tr>
<tr>
<td><strong>Supported Native Formatting Features</strong></td>
<td>7-40</td>
</tr>
<tr>
<td>General Features</td>
<td>7-40</td>
</tr>
<tr>
<td>Alignment</td>
<td>7-41</td>
</tr>
<tr>
<td>Tables</td>
<td>7-41</td>
</tr>
<tr>
<td>Date Fields</td>
<td>7-44</td>
</tr>
<tr>
<td>Multicolumn Page Support</td>
<td>7-45</td>
</tr>
<tr>
<td>Background and Watermark Support</td>
<td>7-46</td>
</tr>
<tr>
<td><strong>Template Features</strong></td>
<td>7-48</td>
</tr>
<tr>
<td>Page Breaks</td>
<td>7-48</td>
</tr>
<tr>
<td>Initial Page Number</td>
<td>7-49</td>
</tr>
<tr>
<td>Last Page Only Content</td>
<td>7-50</td>
</tr>
<tr>
<td>End on Even or End on Odd Page</td>
<td>7-53</td>
</tr>
<tr>
<td>Hyperlinks</td>
<td>7-53</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>7-56</td>
</tr>
<tr>
<td>Generating Bookmarks in PDF Output</td>
<td>7-56</td>
</tr>
<tr>
<td>Check Boxes</td>
<td>7-57</td>
</tr>
<tr>
<td>Drop Down Lists</td>
<td>7-58</td>
</tr>
<tr>
<td><strong>Conditional Formatting</strong></td>
<td>7-61</td>
</tr>
<tr>
<td>If Statements</td>
<td>7-62</td>
</tr>
<tr>
<td>If Statements in Boilerplate Text</td>
<td>7-62</td>
</tr>
<tr>
<td>If-then-Else Statements</td>
<td>7-63</td>
</tr>
<tr>
<td>Choose Statements</td>
<td>7-64</td>
</tr>
<tr>
<td>Column Formatting</td>
<td>7-65</td>
</tr>
<tr>
<td>Row Formatting</td>
<td>7-68</td>
</tr>
<tr>
<td>Cell Highlighting</td>
<td>7-70</td>
</tr>
<tr>
<td><strong>Page-Level Calculations</strong></td>
<td>7-72</td>
</tr>
<tr>
<td>Displaying Page Totals</td>
<td>7-72</td>
</tr>
<tr>
<td>Brought Forward/Carried Forward Totals</td>
<td>7-74</td>
</tr>
<tr>
<td>Running Totals</td>
<td>7-78</td>
</tr>
<tr>
<td><strong>Data Handling</strong></td>
<td>7-80</td>
</tr>
<tr>
<td>Sorting</td>
<td>7-80</td>
</tr>
<tr>
<td>Checking for Nulls</td>
<td>7-80</td>
</tr>
<tr>
<td>Regrouping the XML Data</td>
<td>7-81</td>
</tr>
<tr>
<td><strong>Using Variables</strong></td>
<td>7-87</td>
</tr>
<tr>
<td><strong>Defining Parameters</strong></td>
<td>7-88</td>
</tr>
<tr>
<td><strong>Setting Properties</strong></td>
<td>7-90</td>
</tr>
</tbody>
</table>
8 Extended Function Support in RTF Templates
   Extended SQL and XSL Functions................................................................. 8-1
   XSL Equivalents..................................................................................... 8-15
   Using FO Elements................................................................................ 8-17

9 Translating Reports
   Template Translations......................................................................... 9-1
   Report File Translations..................................................................... 9-8

10 Creating a PDF Template
   Overview............................................................................................ 10-1
      Supported Modes........................................................................ 10-1
   Designing the Layout........................................................................ 10-2
   Adding Markup to the Template Layout............................................. 10-3
      Creating a Placeholder.............................................................. 10-4
      Defining Groups of Repeating Fields............................................ 10-7
   Adding Page Numbers...................................................................... 10-9
   Performing Calculations................................................................. 10-13
   Completed PDF Template Example.................................................. 10-14
   Runtime Behavior............................................................................. 10-15
   Creating a Template from a Predefined PDF Form.......................... 10-17
   Adding or Designating a Field for a Digital Signature..................... 10-18
11 Creating Flash Templates

Introduction............................................................................................................................. 11-1
Building a Flash Template................................................................................................. 11-2

12 Creating an eText Template

Introduction............................................................................................................................. 12-1
Outbound eText Templates................................................................................................. 12-2
  Structure of eText Templates......................................................................................... 12-2
  Constructing the Data Tables...................................................................................... 12-6
    Command Rows........................................................................................................ 12-6
    Structure of the Data Rows.................................................................................... 12-12
  Setup Command Tables............................................................................................ 12-16
  Expressions, Control Structures, and Functions.................................................... 12-27
  Identifiers, Operators, and Literals........................................................................ 12-30

13 Setting Runtime Configuration Properties

Setting Runtime Properties............................................................................................... 13-1
Defining Font Mappings.................................................................................................. 13-15

A Supported XSL-FO Elements

Supported XSL-FO Elements......................................................................................... A-1

B Converting Reports from Oracle Reports to Oracle BI Publisher

Overview............................................................................................................................ B-1
Prerequisites.................................................................................................................. B-2
Running the Conversion Utility................................................................................... B-3
Uploading the PL/SQL Package to the Database....................................................... B-5
Moving Converted Reports to the Oracle BI Publisher Reports Repository........... B-5
Testing and Editing Converted Reports..................................................................... B-6

Index
Preface

Intended Audience

Welcome to Release 10.1.3.4 of the Oracle Business Intelligence Publisher Report Designer’s Guide.

This guide is intended for users who will use Oracle Business Intelligence Publisher Enterprise to perform one or all of the following:

- View and Schedule reports
- Design report layouts
- Develop report queries and data models
- Translate reports

For users who will be developing report queries and data models, knowledge of SQL or your data source is assumed.

For users who will be designing report layouts, some experience with the medium of your template type is assumed (for example, Microsoft Word, Adobe Acrobat, or Adobe Flexbuilder). If you are designing advanced report layouts, you may benefit by using an XSL reference.

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

TTY Relay Access to Oracle Support Services

To reach AT&T Customer Assistants, dial 711 or 1.800.855.2880. An AT&T Customer Assistant will relay information between the customer and Oracle Support Services at 1.800.223.1711. Complete instructions for using the AT&T relay services are available at http://www.consumer.att.com/relay/tty/standard2.html. After the AT&T Customer Assistant contacts Oracle Support Services, an Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service
Documentation Accessibility

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

Accessibility of Code Examples in Documentation

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

Accessibility of Links to External Web Sites in Documentation

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

Structure

1 Introduction
2 Getting Started
3 Viewing and Scheduling Reports
4 Creating a New Report
5 Defining the Data Model for Your Report
6 Building a Data Template
7 Creating an RTF Template
8 Extended Function Support in RTF Templates
9 Translating Reports
10 Creating a PDF Template
11 Creating Flash Templates
12 Creating an eText Template
13 Setting Runtime Configuration Properties
A Supported XSL-FO Elements
B Converting Reports from Oracle Reports to Oracle BI Publisher
Related Information Sources

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

http://www.oracle.com/technetwork/community/join/overview/index.html

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

http://www.oracle.com/technology/documentation/

Information specifically related to BI Publisher can be found at:

http://www.oracle.com/technology/documentation/bi_pub.html

Other guides in the Oracle Business Intelligence Publisher documentation set include:

* Oracle Business Intelligence Publisher Administrator’s and Developer’s Guide*
* Oracle Business Intelligence Publisher Installation Guide*
* Oracle Business Intelligence Publisher Certification Guide*
* Oracle Business Intelligence Publisher Release Notes*
* Oracle Business Intelligence Publisher Java API Reference Guide*

All of these guides are available from:

http://www.oracle.com/technology/documentation/bi_pub.html

If your installation is integrated with the Oracle Business Intelligence Enterprise Edition, please see the Oracle Business Intelligence Enterprise Edition documentation available from: http://www.oracle.com/technology/documentation/bi_ee.html
Overview of the Oracle Business Intelligence Publisher Report Designer's Guide

Oracle Business Intelligence Publisher is a reporting and publishing application that enables you to extract data from multiple data sources, create a template to lay out the data in a report, and publish the report to numerous output formats. BI Publisher also enables you to schedule reports and deliver the reports to any delivery channel required by your business.

This guide is for report consumers and report designers.

Overview for Report Consumers

A report consumer may perform the following tasks:

- Run and view reports from the BI Publisher reports repository
- Schedule reports to run at selected intervals and to be delivered via any channel set up by your administrator
- Create an ad hoc analysis of report data using BI Publisher’s Analyzer feature
- Open and manipulate reports in Microsoft Excel using BI Publisher’s Analyzer for Microsoft Excel

The topics in this book that describe performing report consumer tasks are:

- Navigating the Reports Repository, page 3-1
- Viewing a Report Through the Report Viewer, page 3-2
- Using the Online Analyzer, page 3-6
• Using the Analyzer for Microsoft Excel, page 3-11
• Scheduling a Report, page 3-15
• Scheduling a Report To Be Burst, page 3-20
• Managing Your Scheduled Reports, page 3-24
• Viewing Report History and Saved Output, page 3-25

**About Integration with Oracle Enterprise Performance Management Workspace**

Your work environment may be integrated with Oracle Enterprise Performance Management (EPM) Workspace. If so, the way you navigate to BI Publisher tasks will be different. See Using Oracle BI Publisher in Oracle Enterprise Performance Management Workspace, page 3-20 for more information.

**About Integration with Oracle Smart Space, Fusion Edition**

Your work environment may be integrated with Oracle Smart Space, Fusion Edition. If so, the way you navigate to BI Publisher tasks using this tool will be different. See Using Oracle BI Publisher in Oracle Smart Space, Fusion Edition, page 3-22.

**Overview for Report Designers**

A report consists of a data model, a layout, and a set of properties. A report designer may perform the following tasks:

• Create the report data model, including parameters and lists of values, using BI Publisher’s report editor.

• Design the layout template for the report. The layout can be created using different tools depending on your output requirements, including Microsoft Word, Adobe Acrobat, Microsoft Excel, and Adobe Flexbuilder.

• Set runtime configuration properties for the report.

• Define a bursting control file for reports that are to be burst.

• Enable translations for a report.

**About the Data Source Types**

BI Publisher relies on XML data to format and publish your reports. BI Publisher supports multiple methods for retrieving this data for your report.

• SQL Query
Submit a SQL query against any of the transactional databases set up by your Administrator. BI Publisher also provides a Query Builder that enables you to build your SQL query graphically.

- **HTTP (XML Feed)**
  Use an RSS feed off the Web that returns XML.

- **Web Service**
  Supply the Web service WSDL to BI Publisher and then define the parameters in BI Publisher to use a Web service to return data for your report.

- **Data Template**
  The BI Publisher data engine enables you to rapidly generate any kind of XML data structure against any database in a scalable, efficient manner. The data template is the method by which you communicate your request for data to BI Publisher’s data engine.

- **Oracle BI Answers**
  If you have integrated your BI Publisher installation with Oracle Business Intelligence Presentation Services, then you can use the data from an Oracle BI Answers request to create your report. For more information on this integration, see Integrating with Oracle Business Intelligence Presentation Services, *Oracle Business Intelligence Publisher Administrator’s and Developer’s Guide*.

- **Oracle BI Discoverer**
  If you have integrated your BI Publisher installation with Oracle Discoverer, then you can use the data from an Oracle Discoverer worksheet to create your report. For more information on this integration, see Integration with Oracle Business Intelligence Discoverer, *Oracle Business Intelligence Publisher Administrator’s and Developer’s Guide*.

- **File**
  Use an existing XML data file stored in a directory that has been set up by your Administrator.

- **Multidimensional Query**
  Construct a multidimensional (MDX) query against an OLAP database that has been set up by your Administrator.

**About the Template Types**

BI Publisher offers several options for designing templates for your reports. Templates can be in any of the following formats. Note that some formats restrict output types.
• Rich Text Format (RTF)

BI Publisher provides a plugin utility for Microsoft Word that automates template design and enables you to connect to BI Publisher to access data and upload templates directly from your Word session. RTF templates support the most output types, including PDF, HTML, RTF, Excel, PowerPoint, and MHTML.

• Portable Document Format (PDF)

PDF templates are used primarily for using predefined forms as templates for your reports. For example, you can download forms from government Web sites and load them to BI Publisher as report templates. You can also design your own PDF templates using Adobe Acrobat Professional. BI Publisher provides a mapping tool to enable you to map fields from your data source to the form fields in the PDF template. PDF templates only support PDF output.

• Microsoft Excel (XLS)

Use BI Publisher’s Analyzer for Microsoft Excel to download your report data to an Excel spreadsheet. Create a layout for the data in Excel and then upload the spreadsheet back to BI Publisher to use as a template.

• XSL Stylesheet

You can define a template in XSL formatting language. Specify whether your template is for FO, HTML, XML, or Text transformation.

• eText

These are specialized RTF templates used for creating text output for Electronic Data Interchange (EDI) or Electronic Funds Transfer (EFT) transactions.

• Flash

BI Publisher’s support for Flash templates enables you to develop Adobe Flex templates that can be applied to BI Publisher reports to generate interactive Flash output documents.

About Setting Runtime Properties

BI Publisher provides a variety of user-controlled settings that are specified via an easily accessible Runtime Configuration page. These include security settings for individual PDF reports, HTML output display settings, and other output-specific settings. For more information see Setting Runtime Properties, page 13-1.

About Bursting

Using BI Publisher’s bursting feature you can split a single batch report into individual reports to be delivered to multiple destinations. You can apply a different template, output format, delivery method, and locale to each split segment of your report.
Example implementations include:

- Invoice generation and delivery based on customer-specific layouts and delivery preference
- Financial reporting to generate a master report of all cost centers, bursting out individual cost center reports to the appropriate manager
- Generation of payslips to all employees based on one extract and delivered via e-mail

For more information, see Enabling Bursting, page 4-30.

**About Adding Translations**

BI Publisher provides the ability to create an XLIFF file from your RTF templates. XLIFF is the XML Localization Interchange File Format. It is the standard format used by localization providers. Using BI Publisher's XLIFF generation tool you can generate the standard translation file of your RTF template. You can then translate this file (or send to a translation provider). Once translated, the file can be uploaded to the report definition under the appropriate locale setting so that at runtime the translated report will automatically be run for users selecting the corresponding locale. For more information, see Translating Reports, page 9-1.
Accessing Business Intelligence Publisher Enterprise

Logging in with credentials:
1. Navigate to the URL provided by your system administrator.

2. Select the language you prefer for the user interface.

3. Enter your credentials to log in to BI Publisher.

4. Select **Accessibility Mode** if you wish to render the Reports home page in an accessible tree structure.

5. Select **Sign In**.
To view reports, see Viewing and Scheduling Reports, page 3-1.

To set user preferences, see Setting Preferences, page 2-2.

**Logging in as Guest:**
If your site has enabled a Guest user option, a Guest button will display on the log in page.

A Guest user does not require credentials and has privileges only to view reports available in the Guest folder.

1. Select the language you prefer for the user interface.
2. Select Guest.

To view reports, see Viewing and Scheduling Reports, page 3-1.

**Setting Preferences**

Use the Preferences page to set the following:

- Accessibility Mode
- UI Language
- Report Locale
- Report Timezone
- SVG support in HTML
- Report Viewer Height
- Password

Access the Preferences page by selecting the Preferences link from any page within the BI Publisher Enterprise application.

**Accessibility Mode**

Setting this to "On" will display the report catalog in a tree structure that is accessible via keyboard strokes.

**Set UI Language**

The UI language is the language that your user interface displays in. The language that you selected at login will be selected as the default. Choose from the languages that are available for your installation.
Set Report Locale

A locale is a language and territory combination (for example, English (United States) or French (Canada)). BI Publisher uses the report locale selection to determine the following:

- The template translation to apply
- The number formatting and date formatting to apply to the report data

Note that a particular report must have an available template translation for the selected locale. If not, BI Publisher will apply a locale fallback logic to select the template. For more information, see Locale Selection Logic, page 9-7.

The appropriate number and date formatting will be applied independently of the template translation.

Set Report Timezone

Select the timezone to apply to your reports. Reports run by this user will display the time according to the timezone preference selected here. You can override this setting for a particular report from the Schedule Report, page 3-15. Note that the time displayed on the user interface and reflected in report processing times is governed by the BI Publisher server timezone.

Enable SVG for HTML

You can choose to have graphics in your HTML reports displayed using scalable vector graphics (SVG) technology. Your browser may require a plug-in to enable SVG. If so, you will be prompted to download this plug-in the first time you attempt to view an HTML graphic with SVG enabled. If you do not wish to use the SVG plug-in, select No.

Report Viewer Height

This preference sets the height of the report display region in the View Report page. The default is 500 pixels. You can modify this value to best suit your report viewing device.

Set Your Password

To change your password, select the Account tab of the Preferences page. Enter your current password then your new password as prompted.
Navigating the Reports Repository

The Reports home page offers different functionality depending on your user permissions.

To view a report

1. Navigate to the report.

   The Reports home page displays two main reports folders.

   - **Shared Folders** contains the reports and folders you have been granted access to based on your role

   - **My Folders** contains the reports and folders your administrator has assigned to you and the reports you have created (if you have the BI Publisher Developer or Administrator role).

Each folder displays the first three items (reports or folders) contained in the folder. To see additional items contained in a folder, either select the folder name, or select the **more** link.
Viewing a Report Through the Report Viewer

1. From the Reports home page, select the report name; or, from the Folder view, select the View link for the report. This will run the report using the default options and display it in your browser.

   **Note:** Some reports may not allow online execution. For these reports, the View link will not display. Select Schedule to schedule a report run. See Scheduling a Report, page 3-15.
Folder View

From the Folder view, select the report name or the View link to run the report and view online.

Depending on the report definition and your user permissions, you may be presented with the following options:

- **Change parameter values** - if the report includes parameters, these are presented on the View page. To display the report with new parameter values, enter the values and select View.
- **Change the report template** - if multiple templates are available they will be displayed in the Template list. Select a new template, then select View.

- **Change the output type** - if multiple output types are available, select the desired output type from the list and select View. The output will be rendered in your browser. Possible values are:
  - HTML - (Hypertext Markup Language) formats the report for browser viewing.
  - PDF - (Portable Document Format) - opens the report in Adobe Acrobat reader.
  - RTF - (Rich Text Format) will open your report in Microsoft Word.
  - Excel - choose this output type if you have Microsoft Excel 2003 or later installed. This option generates an MHTML format file that can be opened in Excel 2003 or later. This option supports embedded images, such as charts and logos.
  - Excel2000 - choose this output type if you have Microsoft Excel 2000 or
2002. Note that although Excel2000 is a new choice, it is actually the Excel option that was available in previous releases. This option generates HTML that can be opened in Excel. It does not support embedded images.

- PowerPoint - requires Microsoft PowerPoint 2003 or later.

- MHTML - (Mime HyperText Markup Language) this format enables you to save a Web page and its resources as a single MHTML file (.mht), in which all images and linked files will be saved as a single entity. Use this option if you want to send or save HTML output and retain the embedded images and stylesheet formatting.

- CSV - displays the data in comma separated value format. The data must be in a simple <rowset>/<row> structure.

- Data - displays the report data as XML.

- Flash - displays output for Flash templates. You must have the Adobe Flash Player Plug-in installed for your Web browser.

- **Export the report** - select the Export button to export the report to the default application for its output type (for example: Adobe Acrobat for pdf output or Microsoft Excel for excel output).

- **Send the report** - select the Send button to invoke the Destination dialog. Select the delivery method (Email, Printer, Fax, FTP or Web Folder) and enter the appropriate information for your choice.

  **Note:** To Send a report to the Printer or Fax, you must first change the output type to PDF and select View. Then select Send.

- **Link to this report** - enables you to copy a link to the report you are currently viewing. When you select one of the options, a dialog is invoked that displays the URL to the report. You can choose from the following modes:
  - Current Page - displays the link to the current page as shown.
  - No header - displays the URL to the current report without the logo, tabs, or navigation path.
  - No parameters - displays the URL to the current report without the header or any parameter selections. The controls, such as Template selection, View, Export, and Send will still be available.
• Document - displays the URL to the current report document only. No other page information or options will be displayed.

   Note: Link to this Report is a report property that can be disabled by the report designer. See Using the Report Editor, page 4-3 for information on setting this property.

Access to the following functions must be granted by the System Administrator and may not be available to all users:

• Schedule a report - see Scheduling a Report, page 3-15.

• Invoke Analyzer - see Using the Online Analyzer, page 3-6.

• Invoke Excel Analyzer - see Using the BI Publisher Analyzer for Excel, page 3-11.

Using the Online Analyzer

   Note: Your system administrator must assign you access to this feature.

The online Analyzer enables you to create ad hoc analyses of your data by quickly dragging and dropping your report data elements into a cross tab structure. You can then save this interactive analysis as a template for the report so that you do not have to rebuild your analysis each time you view the report. The interface enables you to easily rearrange and pivot your data by dragging items to different row, column or summary positions.

You can filter the data displayed in your pivot table by defining page-level data items. Drag and drop the desired field to the Page item area and then choose from the values that immediately populate the list.

After selecting all the data items for the table, choose whether to view the Sum, Average, or Count of the data items.

This topic contains two procedures:

• Creating an Ad Hoc Analysis, page 3-7

• Saving Your Analysis as a Template, page 3-10

The Online Analyzer Toolbar

The Analyzer toolbar enables you to perform the following functions:
• **New** - if you are currently viewing your report data through an analyzer template or ad hoc table you have created, you can click **New** to be presented with a clean crosstab structure to build a new analysis. If you want to save the existing analysis, be sure to click **Save** before you click **New**.

• **Save** - saves the current analysis as a template.

• **Save As** - saves the current analysis as a new template. Use this option if you have manipulated an existing analyzer template and wish to save your new analysis as a new template.

• **Rename** - enables you to rename the current analyzer template.

• **Delete** - deletes the analyzer template and clears the analyzer interface.

• **Move decimal left** - click once to display additional decimal positions. You can click multiple times.

• **Move decimal right** - click once to remove the display of a decimal position. You can click multiple times.

**Creating an Ad Hoc Analysis:**
The following example displays the usage of the Analyzer with a simple Sales Analysis report:

1. Select the **Analyzer** button from the View Report page.
   
   The Analyzer interface displays the list of data fields on a pane and an empty crosstab structure on the adjacent pane, as shown in the following figure.
2. To filter by CATEGORY_NAME, drag the item to the Page Items region, as shown in the following figure:

Now you can choose a value from the CATEGORY_NAME list to filter the page data:
3. To view product sales by year, drag PRODUCT_NAME into the Row Field area, and drop ORDER_YEAR into the Column Field area. Drop the SALES data into the table body area, as shown in the following figure:

You can now see the calculated sales totals as a sum of the data items.
4. Add the dimension of ORDER_PERIOD to the table by dragging the data item over the ORDER_YEAR. Now you can click the ORDER_YEAR to open it up to display each ORDER_PERIOD total. Click again to close the item and view only the ORDER_YEAR total.

Saving Your Analysis as a Template:
To save your analysis as a template that you can view later:
1. Click **Save**, or if you have manipulated an existing template, click **Save As**.

2. At the prompt, enter a new Template Name. A confirmation message displays to indicate the template has been saved successfully. The analyzer template will be added to the Template list under the Analyzer heading, and the output type will be displayed as Interactive.

3. To view your template next time you view the report, select it from the Template list, as shown in the following figure:

   ![Image of Oracle BI Publisher Enterprise]

   **Using the BI Publisher Analyzer for Excel**

   **Note:** Your system administrator must assign you access to this feature.

   **Prerequisites**
   - Microsoft .NET Framework 2.0
     
     If not installed on your computer, you will be prompted to download it the first time you use the Analyzer for Excel.
   - Microsoft Excel 2000 or later
**Note:** If you are using Windows Vista you must use the Analyzer for Excel link available from the Developer Tools region to install the Analyzer for Excel. If you are using another supported client operating system, you will be prompted to download the Analyzer for Excel the first time you click the Analyzer for Excel button.

**Features**

The Analyzer for Excel offers two modes: Client access enabled and Client access disabled. The mode is set at the report level. The client access enabled mode is the default mode and is described in this section. For information on the client access disabled mode, see the description of the property Disable Client Access from Analyzer for Excel, page 4-5.

The Analyzer for Excel enables you to:

- Export the results of the report query to an Excel spreadsheet.
- Log in to BI Publisher Enterprise from Excel to refresh your data, apply new parameters, and apply a template to the report data.
- Create Excel templates and upload them to the BI Publisher server.
- Access and run your reports from an Excel session.

**Launching the Excel Analyzer**

1. Select the Excel **Analyzer** button from the View report page. You will be prompted to Save or Open the report .xls file.

2. When you open the file, select Enable Macros from the Excel dialog.

   **Note:** You must enable macros to use the Analyzer for Excel.

The report data will render in your Excel application window and the Oracle BI Publisher menu will appear on your Excel menu bar. Note that the data are the results of the report query with no template and default filtering applied.

You can now manipulate the data as you wish in Excel.

If the report has parameters, the parameter names will appear at the top of the sheet, but you must log in to apply new parameter values. See Using the Oracle BI Publisher Menu, page 3-13.
Using the Oracle BI Publisher Menu

You must log in to enable all the menu commands.

Login – allows you to log in to the BI Publisher server.

Note: If you do not have Microsoft .NET Framework 2.0 installed on your computer, you will be prompted to download it. Select the URL and follow the instructions on the Microsoft Web site to download and install .NET. If you do not wish to install .NET, click OK to close the message window.

If this is the first time you have used the Analyzer for Excel, or if you do not have the latest version of Analyzer for Excel, you will be prompted to install the latest version.

Show Report Parameters – displays the updateable parameters and available templates for the report in a toolbar.

Analyzer for Excel Toolbar

To update the data, select a new parameter value then select Refresh Data to refresh the data in the current sheet.

To apply a template, select the template, then select Refresh Formatted Data. This will download the report as HTML into a new worksheet. Select the new worksheet to see the data with the new template applied.

Note: The template you select must have HTML as an available output.

To change the parameters from this worksheet, select the new values, then select Refresh Data, then select Refresh Formatted Data.

Update Excel Template

If you used the Open Template dialog to download a template from the BI Publisher...
server, use this option to upload the updated layout back to the server.

**Add as New Excel Template**

If you used the Open Template dialog to download a template or to open a report from the BI Publisher server, use this option to upload the layout to the server. Also use this option to upload modifications to an existing template under a different name.

Note that if you created any charts on a separate worksheet the charts cannot be scheduled and viewed within BI Publisher Enterprise. Only charts that you create on the same worksheet that is downloaded by the Excel Analyzer can be updated and viewed within the BI Publisher application.

**View Report Online**

Launches the View report page.

**Browse for Reports Online**

This dialog enables you to select reports from the BI Publisher Report Server or the Oracle BI Answers server. You can either load the report data to create a new template, or download an existing template to update it or to use as a starting point for a new template.

When you use the Open Template dialog to initiate the template building process, you can then use the Update Excel Template options from the Oracle BI Publisher Menu to upload the template directly to the appropriate report in the BI Publisher server.

From the Oracle BI Publisher menu, select Open Template.

**Workspace**

The default workspace is the Oracle BI Publisher server; you can also select Oracle BI to connect to the Answers server. Browse the directory structure of the workspace to select the desired report. Select a folder to display its contents in the Reports pane.

**Reports Pane**

The Reports pane lists the reports in the selected directory. Select a report to display the available templates in the Layout Templates pane.

**Open Report**

Loads the XML data of the selected report to the Template Builder.

**Open Layout Template** Downloads and opens the selected template in the Template Builder and loads the XML data.

To start a new template, select <New> from the list of templates then select Open Layout Template; or double-click <New>.

Use the Report Browser’s Up icon to move up the directory structure.

Use the View As menu to view the folder contents as Large Icons, Small Icons, List or Details.

**Preferences** - select your locale and proxy settings if required.
Logging in Through Excel

Once you have installed the Analyzer for Excel, you can log in to the BI Publisher Enterprise server any time from Excel, you do not have to log in to BI Publisher first.

Once you have Excel open, simply select Log in from the Oracle BI Publisher menu. The BI Publisher Enterprise log in screen will prompt you to enter your credentials and to select (or enter) the Report Server URL.

Scheduling a Report

To schedule a report:

1. Select the name of the folder that contains the report to access the Folder view; or, select the report name to View the report.

2. Select the Schedule link.

3. On the Schedule Report page, enter the following:
   - **Report Parameters** (if applicable) - if the report definition includes parameters, select the desired values for this submission.
   - If this report is to be burst, select the Burst Report check box, under Job Properties. For more information on report bursting, see Scheduling a Report to be Burst, page 3-20.
   - **Template** - select the layout template to apply to the report. You can apply one template per job submission. If you selected Burst Report, this option will not be available.
   - **Format** - select the output format. If you selected Burst Report, this option will not be available.
   - **Job Name** - enter a name for your report run.
   - **Report Formatting Locale** - Select the language-territory combination for the report. This field defaults to the Report Locale defined in the user Preferences (see Setting Preferences, page 2-2).

   **Note:** A report must have an available template translation for the selected locale. If not, BI Publisher will apply a locale fallback logic to select the template. For more information, see Locale Selection Logic, page 9-7.

   The appropriate number and date formatting will be applied.
independently of the template translation.

- **Report Formatting Time Zone** - select the time zone that you want use for the published report. The time zone defaults to the time zone of the BI Publisher server.

- **Report Formatting Calendar** - select the calendar to apply to the date.

- **Burst Report** - select this check box if you wish this report run to be burst. See Scheduling a Report to be Burst, page 3-20.

- **Public** - select this check box to make this job available to all users with access to the report. Users with access can view the report from the **History** page.

- **Save data for Republish** - select this check box if you want the XML data from the report run saved.

- **Save Output** - select this check box if you want the report output saved. You must select this option if you want to view your report from the **History** page.

- **Use Unicode (UTF8)**
4. Select a notification option if desired.

If you wish to send a notification by e-mail:

Select the report completion status or statuses for which you want to be notified, and enter a comma-separated list of addresses to which to send the notification.

If you wish to send an HTTP notification:

Select the report completion status or statuses for which you want the notification to be sent. Select the server to which to send the HTTP notification. Enter the user name and password for the server, if required.

Note that the HTTP server must be set up as a delivery option by the Administrator for the HTTP notification to be available. For more information, see Set Up an HTTP Server, Oracle Business Intelligence Publisher Administrator’s and Developer’s Guide.

5. Enter the Time criteria.

   • If you select Run Once, select the Run Date and Run Time.

   • If you select Run Daily/Weekly select the days of the week, the Run Time, Active Start Date to begin the recurring job and the Active End Date to end the recurring schedule.

   • If you select Run Monthly, select the month, the day of the month to run the report, the Run Time, the Active Start Date to begin running the report and the
Active End Date. To select multiple days of the month to run the report, enter each day separated by a comma (example: 1,15,28).

6. Select the Destination mode and enter the appropriate fields for your selection. To deliver via multiple channels, select the Add Destination button and continue adding destinations as needed.

If you do not wish to choose any of these destinations, leave this region blank. Select the Save output check box in the Job Properties region to view the output from the History page. See Viewing Report History, page 3-25.

Note that delivery options must be set up by your Administrator. For more information, see Setting Up Delivery Options, Oracle Business Intelligence Publisher Administrator’s and Developer’s Guide.

- Email - enter multiple e-mail addresses separated by a comma. Enter any Body text that you want to include with the report.
- Printer - Select the Printer Group and the Printer, enter the Number of copies, and select Single sided or Double sided (the printer must support duplex printing for this option to take effect), the optionally select the printer Tray from which to print the report, and the Pages to print if you do not wish to print the entire report.
- Fax - select the Fax server to deliver the report and enter the Fax number to which to send the report.
- FTP
  - FTP Server - select the server to deliver the report.
  - Username - enter a valid username for the server.
  - Password - enter a valid password.
  - Remote Filename - enter the full path to the file on the remote server. (Example: /home/user/mreport.pdf)
  - Use Secure FTP - select the check box to use secure FTP.
- Web Folder
  - Web Folder Server - select the server to deliver the report.
  - Username - enter a valid username for the server.
  - Password - enter a valid password.
- **Remote Filename** - enter the full path to the file on the remote server.
  (Example: /public/myreport.pdf)

7. Select **Submit**. This will invoke the Schedules page where you can monitor your report. See Managing Your Scheduled Reports, page 3-24.

**Scheduling a Report To Be Burst**

If your report has been enabled for bursting, the Schedule Report page will include a **Burst Report** option under the **Job Properties** region. Once you select this option, the Template and Format parameters and the Delivery options for the report run are disabled because these parameters are defined in the delivery dataset defined for the report. See Enabling a Report for Bursting, page 4-30 for more information on bursting set up.

**Using Oracle BI Publisher in Oracle Enterprise Performance Management Workspace**

Oracle Enterprise Performance Management Workspace (Workspace) is a component of Oracle's Hyperion Foundation Services. It is the central Web interface for users to access all Performance Management content and tools. With its ease-of-use and flexibility, Workspace provides users with a "windows-on-the-Web" experience.

Oracle BI Publisher can be integrated into the Oracle Enterprise Performance Management Workspace, Fusion Edition 11.1.1 release.

For information on setting up the integration between BI Publisher and EPM Workspace, see Setting Up Integration with Oracle Enterprise Performance
Management Workspace, Oracle Business Intelligence Publisher Administrator’s and Developer’s Guide. After you configure Oracle BI Publisher to run within EPM Workspace, you can begin using BI Publisher, as described in the following sections.


Launching Oracle BI Publisher in EPM Workspace

To launch Oracle BI Publisher, use the Navigate menu in the upper-left corner of EPM Workspace window.

Select Navigate, then Applications, then Oracle BI Publisher.

When BI Publisher launches, the BI Publisher menu will appear on the EPM Workspace menu bar. The BI Publisher menu contains the following selections:

- Reports
- History
- Schedule
- Admin

These correspond to the Reports, Schedules and Admin tabs in the BI Publisher native user interface. The History option launches the History of Scheduled Reports page. Use the menu to navigate to the desired BI Publisher page.

Specifying Oracle BI Publisher Preferences Within EPM Workspace

To specify preferences for BI Publisher within Workspace, complete the following steps:

1. From the File menu, select Preferences.

2. From the left-hand list, select Oracle BI Publisher.

3. Update the information on items such as your report locale and time zone.

   **Note:** Accessibility Mode and UI Language are inherited from Workspace preference settings and are not updateable from the BI Publisher Preferences dialog. For information on setting the user interface language for Workspace, see the Oracle Enterprise Performance Management Workspace, Fusion Edition Administrator’s Guide.

4. When you are finished, click OK.

For more information on BI Publisher preferences see Setting Preferences, page 2-2.
Performing Oracle BI Publisher Administration Tasks Within EPM Workspace

All BI Publisher Administration tasks can be performed within EPM Workspace. After launching BI Publisher within Workspace, choose **Admin** from the BI Publisher menu. This will launch the BI Publisher Administration interface. For more information about the administration tasks see the *Oracle Business Intelligence Publisher Administrator’s and Developer’s Guide*.

Downloading the Desktop Tools

To download BI Publisher’s desktop tools, the Template Builder for Microsoft Word Add-in and the Analyzer for Microsoft Excel, perform the following:

From the **Tools** menu, select **Install**, then choose Template Builder or Analyzer for Excel.

For more information about the desktop tools, see *Creating an RTF Template Using the Template Builder for Microsoft Word Add-in*, page 4-17 and *Using the BI Publisher Analyzer for Microsoft Excel*, page 3-11.

Running Reports

Navigate to the Reports repository by selecting the **BI Publisher** menu, then **Reports**. Navigate to and select the report to run as you would in the native BI Publisher interface. For more information see *Navigating the Reports Repository and Viewing a Report*, page 3-1.

Viewing Report Schedules and History

To view report schedules:

From the BI Publisher menu, select Schedule. For more information about the Schedules page, see *Managing Your Scheduled Reports*, page 3-24.

To view report history:

From the BI Publisher menu, select History. For more information about the Report History page, see *Viewing Report History and Saved Output*, page 3-25.

Using Oracle BI Publisher in Oracle Smart Space, Fusion Edition

Oracle Smart Space, Fusion Edition is a personalized information delivery solution for Business Intelligence and Enterprise Performance Management that uses configurable desktop gadgets to deliver BI and EPM to every user in the enterprise, utilizing the Windows desktop. In addition, Oracle Smart Space includes a development toolkit for creating additional gadgets that employs common development languages and methodologies and a secure instant messaging system for shared decision making.
You can use gadgets from Oracle Smart Space, Fusion Edition 11.1.1 release to access and interact with the Oracle BI Publisher reports repository. For example, you can launch BI Publisher reports in the Oracle Smart Space Content Viewer and add BI Publisher reports to a Smart Book gadget and a Favorites gadget. You can use Oracle Smart Space Collaborator to share BI Publisher content with other Oracle Smart Space users.

For information on setting up the integration between Oracle BI Publisher and Oracle Smart Space and for information on using Oracle Smart Space, see the *Oracle Smart Space, Fusion Edition User’s Guide*.

**Performing BI Publisher Tasks in Oracle Smart Space**

Once Oracle Smart Space is connected to BI Publisher, you can use Windows Explorer to browse the BI Publisher repository and see the folders and reports that are available to you.

From Windows Explorer, right-click a report name to see the available BI Publisher report functions. The following figure shows the right-click menu launched for a BI Publisher report in Oracle Smart Space:

Following are the BI Publisher report functions available from the Oracle Smart Space right-click menu. Note that to enable some functions in Oracle Smart Space report-level properties must be set in BI Publisher. For more information about setting the report-level properties, see Using the Report Editor, page 4-3.
<table>
<thead>
<tr>
<th>Oracle Smart Space Function</th>
<th>Description</th>
<th>Required Report-Level Properties in BI Publisher</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open</td>
<td>Opens the report with the default template in the Oracle Smart Space Content Viewer. The default view in the Content Viewer does not show the report control region.</td>
<td>Auto Run, Run Report Online</td>
</tr>
<tr>
<td>Open with controls</td>
<td>Select this option to view the Control region of the report. The Control region consists of the Template list, Output type list, and Parameter lists. Note that the property “Show controls” must be enabled for the report in BI Publisher for this option to be active.</td>
<td>Show Controls, Run Report Online</td>
</tr>
<tr>
<td>Open a template</td>
<td>If multiple templates are available you can select this option to choose another template for the report. Once you have selected a template from the list, the submenu will display the available output types for the template (for example, PDF, RTF, HTML, EXCEL). Output types are enabled at the template-level in the BI Publisher report definition. Because the Template list is part of the control region, the property “Show Controls” must be enabled for the report in BI Publisher for this option to be active.</td>
<td>Auto Run, Run Report Online</td>
</tr>
<tr>
<td>Open in BI Publisher</td>
<td>Launches the BI Publisher View Report page. For more information about this page, see Viewing a Report Through the Report Viewer, page 3-2.</td>
<td>None</td>
</tr>
<tr>
<td>View Scheduler</td>
<td>Launches the BI Publisher Schedule Report page. For more information about this page, see Scheduling a Report, page 3-15.</td>
<td>None</td>
</tr>
<tr>
<td>View History</td>
<td>Launches the BI Publisher History page. For more information about the Report History page, see Viewing Report History and Saved Output, page 3-25</td>
<td>None</td>
</tr>
</tbody>
</table>

**Managing Your Scheduled Reports**

The **Schedules** tab displays information about scheduled reports and the **History** of reports that have already run.
**Schedules Page**

Navigate to this page by selecting the **Schedules** tab, and then the Schedules subtab.

- View current schedules for your private, shared, and public reports
- Monitor the status of a submitted report
- Delete a scheduled report
- Suspend/Resume a scheduled report
- View the submission details

**Viewing Report History and Saved Output**

The **History** page displays information about scheduled reports and reports that have already run.
Navigate to this page by selecting the Schedules tab then the **History** subtab. Use this page to:

- View the status of private, shared, and public submitted reports
- View start and end processing times
- Download or view the XML data produced from the report (if you selected **Save Data** for the report)
- Download or view the report document (if you selected **Save output**)
- View report submission details
- Republish the report data using other formats or templates (if you selected **Save Data** for the report)

You can sort the table of reports by Job Name, Status, Username, Scope, Start Time, or End Time by selecting the column heading.
Process Overview

**Note:** You must be assigned the BI Publisher Developer role or BI Publisher Administrator role to create or edit reports.

Creating a new report consists of the following steps:

1. Create the report entry in the desired folder on the Reports page.
2. Open the Report Editor.
3. Specify the general properties for the report.
4. Define the Data Model.
   
   Your report data may come from a SQL query, an HTTP feed, a Web service, an Oracle BI Answers request, a file, an Oracle Discoverer Worksheet, or BI Publisher’s data template.
5. Define the parameters that you want users to pass to the query, and define lists of values for users to select parameter values.
6. Test your data model.
7. Design the layout template.
   
   - If you are designing an RTF template, load the data to the Template Builder for Word. Use the Template Builder in conjunction with the instructions in Creating an RTF Template, page 7-1 to build your report layout.
   
   - If you are designing a PDF template, follow the instructions in Creating a PDF Template, page 10-1 to build your report layout.
• If you are using a predesigned PDF form (such as a government form) follow the instructions under Mapping Data to PDF Form Fields.

• If you are designing a Flash template, follow the instructions in Creating a Flash Template, page 11-1.

• If you are creating an eText template, follow the instructions in Creating an eText Template, page 12-1.

8. Upload your templates to the Report Editor.


10. (Optional) Enable bursting.

11. (Optional) Add translations for your reports. See Translating Reports, page 9-1.

Create the Report Entry and Specify General Properties

   Note: You must be assigned the BI Publisher Developer role or BI Publisher Administrator Role to create or upload reports.

1. Navigate to the folder in which you want the new report to reside.

   To create a new folder for this report, select the Create a new folder link.

2. Select the Create a new report link from the Folder and Report Tasks menu. This will invoke a text box for you to enter the name of your new report.
3. Enter the name for your new report and select Create. This creates the listing for your report within the current folder.

4. Select the Edit link for the new report entry. This invokes the Report Editor.

Using the Report Editor

1. Navigate to the Report Editor by selecting the Edit link for a report.

   From this page you can:
   
   • Define General properties for the report, page 4-3
   
   • Define the Data Model, page 4-6
   
   • Set up a List of Values, page 4-9
   
   • Define Parameters, page 4-9
- Define Layouts, page 4-14
- Set Up Bursting Options, page 4-30
- Generate an XLIFF file, page 9-3

2. Enter the **Report Properties**:
   - **Description** - the description will display beneath the report name within the report folder.
   - **Default Data Source** - select the data source from the list of values. You may define multiple data sources for your report when you define the Data Model. The Default Data Source you select here will be presented as the default for each new data set you define. Select **Refresh Data Source List** to see any new data sources added since your session was initiated.
   - **Parameters per line** - enter the number of parameters that you want to display before creating a second parameter line for the report. The parameter line is displayed in the online report View page and the Schedule page.
   - **Run report online** - disable this property if you do not want users to view this
report in the online Viewer. When disabled, users will be able to Schedule the report only. For most reports you will keep this enabled. Disable it for long-running, batch, or other reports for which online viewing is not appropriate.

- **Show controls** - enable this property to display the control region of the report. Control region consists of the Template list, Output list, and Parameter lists.

- **Show Report Links** - when you view a report in BI Publisher's report viewer, a "Link to this Report" link displays enabling you to copy the URL of the current report. If you do not want to enable users to see and copy the report link, disable this property.

- **Open Links in New Window** - enable this property to open any links contained in the report in a new browser window.

- **Auto Run** - enable this property to automatically run the report when the user selects the report or the View link for the report within the report folder. When Auto Run is not turned on, selecting the report or the View link for the report displays the online Viewer and parameters for the report only. The user must select the View button from the online Viewer to run the report.

- **Enable document cache** - enable this property to cache the report document per user. With document cache enabled, when a user views the report online, the document (data plus layout) will be placed in cache. When the same user uses the online viewer to view the exact same report (same layout, same output type, same parameter selections) the document will be retrieved from cache. The document will remain in cache according to the cache specifications set in the System Maintenance page. See Setting Cache Specifications, Oracle Business Intelligence Publisher Administrator’s and Developer’s Guide. Note that scheduled reports do not use document cache.

- **Disable Client Access from Analyzer for Excel** - this property controls the method by which report data is downloaded to Excel and also impacts the ability to interact with the BI Publisher server from Microsoft Excel. Turning this property on has the following effects:
  - Your report data downloads faster and large data sets are handled more efficiently
  - You do not have to enable macros
  - You can enable your own custom macros
  - You cannot log in or connect to the BI Publisher server from your Microsoft Excel session. Therefore you cannot upload a template directly from Excel,
nor can you update the report parameters or apply a new template.

You would typically consider turning this property on for reports that generate very large data sets that you wish to manipulate in Excel.

The following table details the differences when this property is enabled:

<table>
<thead>
<tr>
<th>Consideration</th>
<th>&quot;Disable Client Access from Analyzer for Excel&quot; property turned on</th>
<th>Native Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>Data is downloaded faster to Excel and large data sets are handled more efficiently</td>
<td>Download is slower and very large data sets can impact the functioning of the Add-in</td>
</tr>
<tr>
<td>Macros</td>
<td>You do not have to enable macros to use the Excel Analyzer in this mode. You can also create your own custom macros to use with the Excel Analyzer.</td>
<td>You must enable macros to use the Excel Analyzer in this mode. Custom macros are not supported in this mode.</td>
</tr>
<tr>
<td>Connection with BI Publisher</td>
<td>No connection after data is downloaded. You cannot upload templates, change parameters, or apply new templates to the data.</td>
<td>You can connect to the BI Publisher server from your Excel session. You can directly upload templates to the report, update the report parameters, and apply new templates from within your Excel session.</td>
</tr>
</tbody>
</table>

For more information about the Analyzer for Excel, see Using the BI Publisher Analyzer for Excel, page 3-11.

3. Select the Save icon to save your report definition.

Defining the Data Model

BI Publisher requires XML data to publish reports. The XML data can come from any of the following sources:

- SQL query against any database you have set up, including the Oracle BI Server. See Defining a SQL Query Data Set Type, page 5-3.

- HTTP (XML feed)
To Define the Data Model:

1. Select Data Model.
This will display the **Main Data Set** list. This list will be empty until you define a data set. To define a data set, select **New**.

- Enter a **Name** and **Type** for the data set. The **Type** can be:
  - SQL Query
  - HTTP (XML Feed)
  - Web Service
  - Data Template
  - Oracle BI Answers
  - Oracle BI Discoverer
  - File
  - MDX Query

**Important:** If your data set is a Web Service or HTTP (XML Feed) you must define any parameters before you define the data set.
2. After you have defined your data sets, select **Data Model**. The data sets that you have defined will now populate the list for **Main Data Set**.

   **Note:** If you are defining multiple data sets from SQL queries, you can combine them into a single data set by selecting Concatenated SQL Data Source. It is strongly recommended that you select **Make row names unique** if you are concatenating datasets.

---

### Adding Parameters and Lists of Values

Add parameters to your report definition to enable your users to interact with the report and specify the data of interest from the data set; or specify hidden parameters to control the data returned to a user from a data set.

**Note:** Parameters are not supported for Oracle BI Answers request data set type.

BI Publisher supports the following parameter types:

- **Text** - allows the user to enter a text entry to pass as the parameter.

- **Menu** - allows the user to pass parameters by making selections from a list of values. This option supports multiple selections, a "Select All" option, and partial page refresh for cascading parameters. Define the properties for the list of values in the report definition. A list of values can contain fixed data that you specify or the list can be created via a SQL query executed against any of the defined data sources.

To add a parameter as a menu, define the list of values first. Then define the parameter and associate it to the list of values. See Adding a List of Values, page 4-9.

- **Date** - allows the user to enter a date as a parameter. Note that the data type must also be "Date" and the format must be Java date format.

- **Hidden** - enables you to pass the default value always, without allowing the user to see or change it.

**Adding a List of Values:**

- Select List of Values and then select the New icon in the toolbar. This will create a **New List of Values** entry.

- Enter a Name for the list and select a Type: SQL Query or Fixed Data.
If you select SQL Query:

- Select a Connection from the data source list.
- Select **Cache Result** if you want the results of the query cached for the report session.
- Enter the SQL query or use the Query Builder. See Using the Query Builder, page 5-4 for information on the Query Builder utility.

If you select Fixed Data:

- Select the **Add** link to add the Label and Value pairs for the LOV.

**Adding Parameters**

Select **Parameters** and then select the **New** icon to define parameters for the report.
• Enter a name **Identifier** and the **Data Type** (String, Integer, Boolean, Date, or Float).

• Enter a **Default Value** for the parameter, if desired. Enter * to pass All as the default.

  **Note:** Using * passes a null, so you must handle the null in your data source. A method to handle the null would be the standard Oracle NVL command, for example:

  ```sql
  where customer_id = nvl(:cstid, customer_id)
  ```

  where cstid is a value passed from the LOV and when the user selects All it will pass a null value.

  If your data source is the Oracle BI Server, use the following macro to handle the null:

  ```{template}
  {$ if ${sYear}='*'$}
  {$elsif ${sYear}='2000' $}
  where Year = :sYear
  {$else $}
  where Year = :sYear
  {$endif$}
  ```

  where Year is a value passed from the LOV and when the user selects All it will pass a null value.
Note that the test operator must be either "= or "!=".

- Select the **Parameter Type**:
  - **Text** - this type allows the user to enter a text entry to pass as the parameter. Enter the Display Label for the parameter and the Text Field Size in characters. You may also enable the following options:
    - **Text field contains comma-separated values** - select this option to enable the user to enter multiple comma-separated values for this parameter.
    - **Refresh other parameters on change** - performs a partial page refresh to refresh any other parameters whose values are dependent on the value of this one.
  - **Menu** - this type presents a list of values to the user. Enter the Display Label and select from the lists you defined in the previous step. You may also enable the following options:
    - **Multiple Selection** - allows the user to select multiple entries from the list.
    - **Can select all** - inserts an "All" option in the list. When the user selects "All" from the list of values, you have the option of passing a null value for the parameter or all list values. Choose **NULL Value Passed** or **All Values Passed**.
    - **Refresh other parameters on change** - performs a partial page refresh to refresh any other parameters whose values are dependent on the value of this one.
  - **Date** - passes a date parameter. If you select a Parameter Type of Date, the Data Type automatically defaults to Date. Enter the following:
    - **Display Label and Text Field Size in characters**.
    - **Date Format String** - enter a format string for the date. It must be a Java data format.
    - **Date From and Date To** - enter the to and from dates that you want to restrict the calendar date picker to.
  - **Hidden** - select this option to pass the default value always, without allowing the user to see or change it.
Adding Layouts to the Report Definition

BI Publisher offers several options for designing templates for your reports. Templates can be in any of the following formats. Note that some formats restrict output types.

- Rich Text Format (RTF)

  RTF is the most common template type. Use Microsoft Word to design the template. Most Microsoft Word formatting features are supported. BI Publisher provides a plugin utility for Microsoft Word that automates template design and enables you to connect to BI Publisher to access data and upload templates directly from your Word session. See Creating an RTF Template, page 7-1.

- Portable Document Format (PDF)

  PDF templates are used primarily for using predefined forms as templates for your reports. For example, you can download forms from government Web sites and load them to BI Publisher as report templates. You can also design your own PDF templates using Adobe Acrobat Professional. BI Publisher provides a mapping tool to enable you to map fields from your data source to the form fields in the PDF template. See Creating a PDF Template, page 10-1.

- Microsoft Excel (XLS)

  Use BI Publisher’s Analyzer for Excel to download your report data to an Excel spreadsheet. Create a layout for the data in Excel and then upload the spreadsheet back to BI Publisher to use as a template. See Using the BI Publisher Analyzer for Excel, page 3-11.

- XSL Stylesheet

  You can define a template in XSL formatting language. Specify whether your template is for FO, HTML, XML, or Text transformation. To add your template, follow the steps in Adding a Layout - General Steps, page 4-14.

- eText

  These are specialized RTF templates used for constructing EDI or EFT transactions. See Creating an eText Template, page 12-1. To add your template, follow the steps in Adding a Layout - General Steps, page 4-14.

- Flash

  BI Publisher’s support for Flash templates enables you to develop Adobe Flex templates that can be applied to BI Publisher reports to generate interactive Flash output documents. See Creating a Flash Template, page 11-1. To add your template, follow the steps in Adding a Layout - General Steps, page 4-14.

- Autogenerate Layout - if your data model type is a SQL query or Data Template, BI
Publisher can autogenerate an RTF template. The autogenerated template constructs a simple table containing all the elements from your data. See Adding a Layout - General Steps, page 4-14.

To add a layout to your report definition, select **Layouts** to specify the layout template for the report. Defining layouts consists of two steps: Upload a template file, and then assign the template file to a Layout definition. If you are connected to BI Publisher through the Template Builder or Excel Analyzer, you can upload the layout file in one step.

**Note:** To build a template for your report, you must have sample data. Once you have defined your query, you can select the View link to generate XML. Select the Export button and save the file to your local directory. If you are building an RTF template or Excel template you can load this data directly to the Template Builder for Word or Excel using BI Publisher’s desktop tools described in the following sections.

### Adding a Layout - General Steps

To add a layout to your report definition, select **Layouts** to specify the layout template for the report. Defining layouts consists of two steps: Upload a template file, and then assign the template file to a Layout definition. If you are connected to BI Publisher through the Template Builder or Excel Analyzer, you can upload the layout file in one step.

See Creating an RTF Template Using the Template Builder for Word, page 4-17.

BI Publisher can also autogenerate a template for you if your data model type is a SQL query or data template. See Autogenerating a Layout, page 4-17.

The general guidelines for uploading and defining the layout for any template type are as follows:

1. **Upload your layout template file.**

   From the BI Publisher Report Editor. Select **Layouts**.

   Use the **Browse** button to locate it in your local file system, then select **Upload**. The template will now appear in the **Manage Template Files** region. You can upload as many templates as you want to make available to this report.
2. Select the **New** icon to create the definition for the new template.
• Enter a **Name** for the layout definition. This name will appear in the Template list on the View report page.

• Select the **Template** file from the list of uploaded templates to correspond to this layout definition.

• Select the appropriate template type you are uploading: RTF, PDF, Excel, XSL, or eText.

• Select the **Output Format** types to allow for this layout.

  If the template type is RTF, you can either select **All Formats** or limit the allowed formats by selecting only those desired.

  All other template types have specific output formats. For these, **All Formats** is automatically selected. The allowed output type for each of the other template types is the same as the template type (example: PDF Templates allow PDF output only).

  **Note:** You can also manage the output types allowed through the Runtime Configuration properties. However, the setting on the report definition will override the configuration. See Setting Runtime Properties, page 13-1.
3. Select Save. The Layout will now appear as an available template when you run the report.

4. Select a Default Template. The Default Template will be used by default by the online viewer and the scheduler unless the user selects another.

**Autogenerating a Layout**

If your data model type is a SQL query or a Data Template, BI Publisher can autogenerate an RTF template for you using the default data model for your report. The autogenerated template will present the elements from your data in a simple table layout.

To autogenerate a layout:

1. From the BI Publisher Report Editor. Select **Layouts**.

2. In the *Auto Generate Layout* region enter a name with the .rtf extension for your layout file (for example: simplelisting.rtf).

3. Follow the instructions starting with Step 2 in the above procedure, Adding a Layout - General Steps, page 4-14 to associate the template file with a layout definition.

**Creating an RTF Template Using the Template Builder for Word**

**Prerequisites:**

- Your report data model has been created and runs successfully.

- Microsoft Word version 2000 or later and Microsoft Windows version 2000 or later are installed on your client.

- The Template Builder has been downloaded and installed on your client.

The Template Builder can be downloaded from the BI Publisher **Folder and Report Tasks** region.
Features of the Template Builder

When you open Microsoft Word after installing the Template Builder you will notice the Oracle BI Publisher menu and the BI Publisher toolbar.
The toolbar and the menu provide two methods of performing many of the same functions, including:

- Insert data fields into your RTF templates
- Insert tables, forms, charts, and crosstabs
- Preview your template in multiple outputs
- Browse and update the content of form fields
- Validate your template
- Perform calculations on fields within the template
- Connect to the Oracle BI Publisher server or the Oracle BI server to retrieve data to build your template
- Publish your template to the Oracle BI Publisher server
- Extract boilerplate text into an XLIFF translation file and test translations

**Building and Uploading Your Template**

You can build and upload your template via a direct connection with the BI Publisher server, or you can build and upload your template in disconnected mode.
Connected Mode

1. Open Microsoft Word.

2. From the Oracle BI Publisher menu, select Log On.

3. Enter your BI Publisher credentials and the URL for the BI Publisher server. (Contact your system administrator if you do not know the URL.)

4. The Open Template dialog presents the same folder structure as your BI Publisher Reports home page. Select the report for which you want to build a template.

5. Select Open Report to load the data to the Template Builder; or double-click <New> in the Layout Templates pane.

   Note that any existing templates will be listed in the Layout Templates pane.

6. Follow the guidelines in the Template Builder online help (from the Oracle BI Publisher menu) to insert data fields and design your template using features such as tables, charts, graphics, and crosstabs. Use Microsoft Word to apply formatting to fonts and other objects in your template.

   For more advanced template options, use the guidelines in Creating an RTF Template, page 7-1.

7. To upload your template to the BI Publisher server and add it to your report definition, select Publish Template As from the Oracle BI Publisher menu.

   If you have not saved your template, you will be prompted to save it in Rich Text Format.

8. Enter a name for your template in the Upload as New dialog. Note that this is the name that appears under Layouts in the Report Editor. This is also the template name that will be displayed whenever the user is presented an option for selecting a
9. (Optional) Limit the output formats for this template.

From the BI Publisher Enterprise interface, open the report in the Report Editor. Under Layouts, select your uploaded template. If you wish to limit the output formats for this report, select only the formats you want to make available.

Disconnected Mode

From the Report Editor:

1. Generate a sample data file.

From the Report Editor or from the Reports page, select View. If no layouts are defined for your report, then the output type will default to xml, otherwise, choose data for the output type. Select Export. Save the results as an XML file to a local directory.

2. Open Microsoft Word with the Template Builder installed.

3. From the Oracle BI Publisher menu select Data and then select Load Sample XML Data. Locate your sample data file in your local directory and select Open. A pop up message will indicate your data has loaded successfully.

4. Follow the guidelines in the Template Builder online help (from the Oracle BI Publisher menu) to insert data fields and design your template using features such as tables, charts, graphics, and crosstabs. Use Microsoft Word to apply formatting to fonts and other objects in your template.

For more advanced template options, use the guidelines in Creating an RTF Template, page 7-1.

5. Upload your layout template file.

Return to your report definition in the BI Publisher Report Editor. Select Layouts. Use the Browse button to locate it in your local file system, then select Upload. The template will now appear in the Manage Template Files region. You can upload as many templates as you want to make available to this report.
6. Select the **New** icon to create the definition for the new template.
• Enter a **Name** for the layout definition. This name will appear in the Template list on the View report page.

• Select the **Template** file from the list of uploaded templates to correspond to this layout definition.

• Select the appropriate template type: RTF or PDF.

• Select the **Output Format** types to allow for this layout.
  
  If the template type is RTF, you can either select **All Formats** or limit the allowed formats by selecting only those desired.
  
  If the template type is PDF, **All Formats** is automatically selected. The only allowed output type for a PDF template is PDF.

  **Note:** You can also manage the output types through the Runtime Configuration Properties. However, the setting on the report definition will override the configuration setting. For more information, see Setting Runtime Properties, page 13-1.
Adding a PDF Template to Your Report

Typically, the source for a PDF template is a predefined form from a third party, such as the government. If form fields have already been defined in the PDF, then you have two options for associating the XML data to the PDF form fields:

- Map the data fields to the form fields in the PDF, using BI Publisher's PDF mapping tool
- Name the fields from your data source to match the names of the form fields.

If you are creating a report to be used exclusively for the preparation of a PDF form, then consider naming the fields in your data according to the form field names in the PDF. If the field names match, no mapping is required.

If the predefined PDF does not have form fields defined, or if you wish to design your own PDF template, then you must use Adobe Acrobat Professional to insert the form fields. You can then either name the fields according to the data source (no mapping will be required) or use BI Publisher’s PDF mapping tool. For information on designing a PDF template and inserting form fields, see Creating a PDF Template, page 10-1.

Determining If a PDF Has Form Fields Defined

If you have the full version of Adobe Acrobat 5.0 or later:

1. Open the file in Adobe Acrobat.

2. Select the Text Field Tool (Adobe Acrobat Professional 6.0 users) or the Form Tool (Adobe Acrobat 5.0 users). This will highlight text fields that have already been defined. If no fields are highlighted then you must add the fields to the PDF. See Adding Markup to the Template Layout, page 10-3 for instructions on inserting PDF form fields.

The following figure shows a sample PDF form opened in Adobe Acrobat Professional 6.0. The Text Field Tool has been selected to display all the available form fields.
If you do not have the full version of Adobe Acrobat 5.0 or later:

1. Follow the instructions in Adding a Predefined PDF Form as a Template, page 4-25.

2. If no highlighted fields display for mapping, or you cannot select a field, then you must add them before you can use BI Publisher’s mapping tool. Adding form fields requires Adobe Acrobat 5.0 or later, or Adobe Acrobat Professional 6.0 or later. For more information, see Creating a PDF Template, page 10-1.

Adding a Predefined PDF Form as a Template

Prerequisites:

- A report data model defined in BI Publisher.
- A PDF document with form fields defined.
- Adobe Acrobat Reader installed as a Web browser plugin. Recommended version is Adobe Acrobat Reader 7.0 or later. (You can use Acrobat Reader 6 if English is the only language required for your site.)

1. From the Report Editor, select Layouts.
2. Upload the PDF template file.

   From the **Manage Template Files** region, select **Browse** to locate the PDF file, and then select **Upload**.


   From the Report Editor or from the Reports page, select **View**. If no layouts are defined for your report, then the output type will default to xml, otherwise, choose **data** for the output type. Select **Export**. Save the results as an XML file to a local directory.

4. Upload the sample data file.

   From the Report Editor, **Layouts** pane, in the **Sample Data** region, browse for and upload your sample data file.

5. Map the PDF form fields.

   Once you have uploaded your template and sample data, the **Map Form Fields** button will become enabled.
6. Select Map Form Fields.

The BI Publisher mapping tool will launch in a separate browser window.

Note that as you mouse over the fields, the name of the field in the PDF form will display.
7. Click in the field on the PDF form that you want to map data to.

A second window will launch, displaying the field names from the sample data that you loaded. Note that the form field selected is shown at the top of the dialog. If the field is already mapped, the dialog will display the name of the data field that it is currently mapped to. In the figure below, ANNUAL_SALARY is the name of the selected form field. It is shown as being mapped to ANNUAL_SALARY in the data (ANNUAL_SALARY = ANNUAL_SALARY).
8. Select the field from the Form Field Mapping dialog and then click Select. This will complete the mapping for the field.

9. Repeat the selection process for each field that you want to map from the PDF template.

10. To see a preview of your template with the sample data mapped to the fields, select Show Preview.

11. When you have mapped all fields, select Submit to save your mapping file.

   Note that the PDF mapping file is saved in the report definition as a .map file.

12. Select the New icon to create the definition for the new template.

   • Enter a Name for the layout definition. This name will appear in the Template list on the View report page.

   • Select the Template file from the list of uploaded templates to correspond to this layout definition.

   • Select the appropriate template type: PDF.

   • The Output Format for PDF templates defaults to All Formats and does not allow update. PDF output is the only allowed output type for PDF templates.
Enabling Bursting

Using BI Publisher’s bursting feature you can split a single report based on a key in the report data and deliver the report based on a second key in the report data. Driven by the delivery key, you can apply a different template, output format, delivery method, and locale to each split segment of your report. Example implementations include:

- Invoice generation and delivery based on customer-specific layouts and delivery preference
- Financial reporting to generate a master report of all cost centers, bursting out individual cost center reports to the appropriate manager
- Generation of payslips to all employees based on one extract and delivered via e-mail

Enabling a Report for Bursting

**Prerequisite:** A report defined in BI Publisher. The report data must contain an element by which the report will be split and an element by which the report will be delivered.

Enabling a report for bursting consists of the following steps:

- Open the report in Edit mode.
- Select **Bursting** under the report definition.
- Select the **Enable Bursting** check box.
- Select the **Split By** and **Deliver By** elements.
  
  The **Split By** element is the data element from the report file that you wish to split the report by. For example, to split a batch of invoices by each invoice, you may use an element called CUSTOMER_NAME.

  The **Deliver By** element is the data element from the report file by which to determine the delivery method. In the invoice example, it is likely that each invoice will have delivery criteria determined by customer, therefore the Deliver By element may be CUSTOMER_ID.

- Select the data source for the delivery XML.
  
  The delivery XML can be sourced from the same data source as the main data set, or it can be generated from a different data source.

- Enter the SQL query to build the delivery XML. See Defining the Delivery Data Set, page 4-31 for details.
Defining the Delivery Data Set

Based on the SQL query that you provide on the Bursting criteria page of the Report Editor, BI Publisher will build the delivery XML data set. The delivery XML data set contains the information to deliver your burst report appropriately to each recipient. The delivery data in this XML document is used as a mapping table for each Deliver By element. The structure of the delivery XML is as follows:

```xml
<ROWSET>
  <ROW>
    <KEY></KEY>
    <TEMPLATE></TEMPLATE>
    <TEMPLATE_FORMAT></TEMPLATE_FORMAT>
    <LOCALE></LOCALE>
    <OUTPUT_FORMAT></OUTPUT_FORMAT>
    <DEL_CHANNEL></DEL_CHANNEL>
    <PARAMETER1></PARAMETER1>
    <PARAMETER2></PARAMETER2>
    <PARAMETER3></PARAMETER3>
    <PARAMETER4></PARAMETER4>
    <PARAMETER5></PARAMETER5>
    <PARAMETER6></PARAMETER6>
    <PARAMETER7></PARAMETER7>
    <PARAMETER8></PARAMETER8>
    <PARAMETER9></PARAMETER9>
    <PARAMETER10></PARAMETER10>
  </ROW>
</ROWSET>
```

where

- **KEY** is the Delivery key and must match the **Deliver By** element. The bursting engine uses the key to link delivery criteria to a specific section of the burst data.

- **TEMPLATE** - is the name of the Layout template to apply. Note that the value is the Layout name (for example, "Invoice"), not the template file name (for example, invoice.rtf).

- **TEMPLATE_FORMAT** - is the format of the layout template. Valid values are:
  - RTF
  - PDF
  - ETEXT
  - XSL_FO

- **LOCALE** - is the template locale, for example, "en-US".

- **OUTPUT_FORMAT** - is the output format. Valid values are: for example: pdf, html, excel.
- HTML
- PDF
- RTF
- EXCEL

- DEL_CHANNEL - is the delivery method. Valid values are:
  - EMAIL
  - FAX
  - FILE
  - FTP
  - PRINT
  - WEBDAV

- Delivery parameters by channel. The delivery parameters by channel are defined in the following table:

<table>
<thead>
<tr>
<th>Parameter Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>Email</td>
</tr>
</tbody>
</table>

Note that if your output format is pdf, you must set this parameter to "true" to attach the pdf to the email.
<table>
<thead>
<tr>
<th>Channel</th>
<th>Parameter 1</th>
<th>Parameter 2</th>
<th>Parameter 3</th>
<th>Parameter 4</th>
<th>Parameter 5</th>
<th>Parameter 6</th>
<th>Parameter 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Printer</td>
<td>Printer Group</td>
<td>Printer</td>
<td>Number of copies</td>
<td>Sides</td>
<td>Tray</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fax</td>
<td>Fax server Name</td>
<td>Fax</td>
<td>Number</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WEBDAV</td>
<td>Server Name</td>
<td>Username</td>
<td>Password</td>
<td>Remote Directory</td>
<td>Remote File Name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>File</td>
<td>Directory</td>
<td>File Name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FTP</td>
<td>Server Name</td>
<td>Username</td>
<td>Password</td>
<td>Remote Directory</td>
<td>Remote File Name</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bursting Example**

**Example**
The following example shows bursting enabled for a report based on the Split By key CUSTOMER_NAME and the Deliver By key CUSTOMER_ID.
The report will be burst and delivered via e-mail. The template, template format, locale, output format, delivery channel, and customer e-mail address are all specified in elements from the delivery data source and will be returned by the query. The SQL to generate the delivery XML for this example is as follows:

```sql
select distinct
CUSTOMER_ID KEY,
CST_TEMPLATE TEMPLATE,
TMPL_TYPE TEMPLATE_FORMAT,
CST_LOCALE LOCALE,
CST_FORMAT OUTPUT_FORMAT,
CST_DEL_CHAN DEL_CHANNEL,
CST_EMAIL PARAMETER1,
'accounts.receivable@oracle.com' PARAMETER2,
'bip-collections@oracle.com' PARAMETER3,
'Your Invoices' PARAMETER4,
'Hi'||CUST_FIRST_NAME||chr(13)|| 'Please find attached your invoices.' PARAMETER5,
'true' PARAMETER6,
'donotreply@nowhere.com' PARAMETER7
from customers
```

For information on running the report, see Scheduling a Report to Be Burst, page 3-20.

**Accessing Reports via a URL**

This section describes how to call a BI Publisher report via a URL from another application, for example from a portal or from an Application Express application.
Security Considerations

In the BI Publisher security model, reports are placed in folders and those folders are then secured to a role and a role assigned to a user. For a user to successfully access the report, you must ensure that the user is credentialed within BI Publisher to see it. There are two options for this:

- Use the Guest folder

  Enable the Guest folder via the Security Configuration tab of the Security Center page (for more information see Allowing Guest Access, Oracle Business Intelligence Publisher Administrator’s and Developer’s Guide). Any report in this folder is open to all users to see and run. Use this option if the report does not contain sensitive data.

- Use SSO

  If both the calling application and BI Publisher are configured as partner applications in an SSO server, you can call any report via a URL and as long as the user has rights to see or run the report, then BI Publisher will render it without the need for the user to log in. For more information on setting up security options, see Defining a Security Model, Oracle Business Intelligence Publisher Administrator’s and Developer’s Guide.

Building the URL

The basic URL for a report is as follows:

```
```

where

- `server:port` - is the name of the server and port number where BI Publisher is running
- `xmlpserver` - is a required string (the name of the application)
- `ReportDirectory` - is the folder path to the report
- `ReportName.xdo` - is the name of the report with the `.xdo` extension.

**Important:** On the BI Publisher server, a report resides in a folder named for the report. For example, assume you have a report called Salary Report. On your BI Publisher desktop it is located in a folder of reports called Executive. Within Executive, it is located in a folder called Private. The path to this report would therefore be

```
Executive/Private/Salary+Report
```

Note that you must replace a space in the folder or report name with the `+` character.

```
ReportName.xdo - is the name of the report with the .xdo extension.
```
This will render the complete report inside the BI Publisher page with all the report controls. The default template, output and parameters will be used to render the report. For example:


server:port - xdopf.us.oracle.com:9999
xmlpserver
ReportName.xdo - Salary+Report.xdo

Specifying Parameters in the URL

If you want to specify parameters for your output report, such as the template, the output format, and any parameters defined for the report, you can add name/value pairs to the URL. The easiest way to generate the URL is to use the Export function from the BI Publisher View Report page. The URL generated will look similar to the basic URL described above, but the name/value pairs will be added.

For example:


The URL components through the report name are described in the previous section. The URL after the report name consists of:

_xpf=&_xpt=1&_xdo=%2FExecutive%2FEmployee+Salary+Report.xdo&dept=10&_xt=Standard&_xf=html

Note the following standard URL syntax:

? - denotes the first parameter
& - denotes each additional parameter

The BI Publisher parameters are as follows:

_xpf - required string for internal use

_xpt - defines whether to render the report in the full BI Publisher window (as above), or to render just the report document. Valid values are

- 0 - uses the BI Publisher window
- 1 - renders just the document

_xdo - (optional) provides the path to the current report
depth - this is a parameter specific to the report as defined in the report definition. In this case the department for the data. Notice it takes the department ID. The parameter definition is to show the user the department name and then pass the ID to the query.
You can have multiple parameters and their values in the URL.

_xt - this controls the template to be used. This is the template name, not the template file name. In this case, the template name is "Standard".

_xt - this controls the format of the output to be generated. Valid values are same as for the report: pdf, html, excel, rtf, or data.
Defining the Data Model for Your Report

This chapter covers the following topics:

- Introduction
- About the Data Model Options
- Defining a SQL Query Data Set Type
- Defining an HTTP Data Set Type
- Defining a Web Service Data Set Type
- Defining a Data Template Data Set Type
- Defining an Oracle BI Answers Request Data Set Type
- Defining an Oracle BI Discoverer Data Set Type
- Defining a File as a Data Set Type
- Defining an MDX Query Data Set Type

Introduction

BI Publisher relies on XML data to format and publish your reports. BI Publisher supports multiple methods for retrieving this data for your report. Moreover, you can combine data from different sources into a single report.

About the Data Model Options

BI Publisher supports the following data model types:

- SQL Query, page 5-3

Submit a SQL query against any of the transactional databases set up by your Administrator. BI Publisher also provides a Query Builder that enables you to build your SQL query graphically.
• **HTTP (XML Feed), page 5-10**
  Use an RSS feed off the Web that returns XML.

• **Web Service, page 5-11**
  Supply the Web service WSDL to BI Publisher and then define the parameters in BI Publisher to use a Web service to return data for your report.

• **Data Template, page 5-18**
  The BI Publisher data engine enables you to rapidly generate any kind of XML data structure against any database in a scalable, efficient manner. The data template is the method by which you communicate your request for data to BI Publisher’s data engine.

• **Oracle BI Answers, page 5-18**
  If you have integrated your BI Publisher installation with Oracle Business Intelligence Presentation Services, then you can use the data from an Oracle BI Answers request to create your report. For more information on this integration, see Integrating with Oracle Business Intelligence Presentation Services, *Oracle Business Intelligence Publisher Administrator’s and Developer’s Guide*.

• **Oracle BI Discoverer, page 5-20**
  If you have integrated your BI Publisher installation with Oracle Discoverer, then you can use the data from an Oracle Discoverer worksheet to create your report. For more information on this integration, see Integration with Oracle Business Intelligence Discoverer, *Oracle Business Intelligence Publisher Administrator’s and Developer’s Guide*.

• **File, page 5-22**
  Use a pregenerated XML data file stored in a directory that has been set up by your Administrator.

• **MDX Query, page 5-22**
  Construct a multidimensional (MDX) query against an OLAP database that has been set up by your Administrator.
Defining a SQL Query Data Set Type

1. Select the **Data Source** for this data set. Select the **Default Data Source** (defined in the Report Properties) or select a new data source from the list.

2. Select the **Cache Result** box if you wish to cache the results of the query for your session.

   By caching the results of the query, multiple templates can be applied to these results without requerying the data. This will enhance online performance. However, if the data is updated during the session, the user cannot view the new data via the View report page until the cache is cleared.

   **Note:** You can control the cache expiration time and the cache size through the configuration settings. See Setting Server Configuration Options, *Oracle Business Intelligence Publisher Administrator's and Developer's Guide* for more information.

3. Enter the SQL query or select **Query Builder**. See Using the Query Builder, page 5-4 for information on the Query Builder utility.
Using the Query Builder

About Query Builder

Use the Query Builder to build SQL queries without coding. The Query Builder enables you to search and filter database objects, select objects and columns, create relationships between objects, and view formatted query results with minimal SQL knowledge.

The Query Builder page is divided into three sections:

- Object Selection pane contains a list objects from which you can build queries. Only objects in the current schema display.

- Design pane displays selected objects from the Object Selection pane.

- Output pane allows you to create conditions, view the generated SQL, or view query results.

Understanding the Query Builder Process

To build a query, perform the following steps:

- Select objects from the Object Selection pane.

- Add objects to the Design pane and select columns.
• Optional: Establish relationships between objects.

• Optional: Create query conditions.

• Execute the query and view results.

**Using the Object Selection Pane**

In the Object Selection pane you can select a schema and search and filter objects.

To hide the Object Selection pane, select the control bar located between it and the Design pane. Select it again to unhide it.

**Selecting a Schema**

The Schema list contains all the available schemas in the data source. Note that you may not have access to all that are listed.

**Searching and Filtering Objects**

Use the Search field to enter a search string. Note that if more than 100 tables are present in the data source, you must use the Search feature to locate and select the desired objects.

**Selecting Objects**

The Object Selection pane lists the tables, views, and materialized views from the selected schema (for Oracle databases, synonyms are also listed). Select the object from the list and it displays on the Design pane. Use the Design pane to identify how the selected objects will be used in the query.

**Supported Column Types**

Columns of all types display as objects in the Design pane. Note the following column restrictions:

• Each can select no more than 60 columns for each query.

• Only the following column types are selectable:
  
  • VARCHAR2, CHAR
  
  • NUMBER
  
  • DATE, TIMESTAMP
  
  • BLOB

  **Note:** The BLOB must be XML or an image. When you execute the query in the Query Builder, the BLOB will not display in the Results pane, however, the query will be constructed correctly when saved to the Report Editor.

• XMLType
**Note:** When you execute the query in the Query Builder, the XMLType will display as null. When you save the query to the Report Builder, you must add the function (such as getClobval()) to extract the XML from the type.

**Adding an Object to the Design Pane**

1. Select an object.

   The selected object displays in the Design pane. An icon representing the datatype displays next to each column name.

2. Select the check box for each column to include in your query.

   When you select a column, it appears on the **Conditions** tab. Note that the **Show** check box on the **Conditions** tab controls whether a column is included in query results. Be default, this check box is selected.

   To select the first twenty columns, click the small icon in the upper left corner of the object and then select **Check All**.

3. To execute the query and view results, select **Results**.

   **Tip:** You can also execute a query using the key strokes CTRL + ENTER.

**Resizing the Design and Results Pane**

As you select objects, you can resize the Design and Results panes by selecting and dragging the gray horizontal rule dividing the page.

**Removing or Hiding Objects in the Design Pane**

To remove an object, select the **Remove** icon in the upper right corner of the object.

To temporarily hide the columns within an object, click the **Show/Hide Columns** icon.

**Specifying Query Conditions**

Conditions enable you to filter and identify the data you want to work with. As you select columns within an object, you can specify conditions on the Conditions tab. You can use these attributes to modify the column alias, apply column conditions, sort columns, or apply functions.

When you select a column to include in your query, it appears as a separate row in the Output pane. The following table describes the attributes available on the **Conditions** tab:
<table>
<thead>
<tr>
<th>Condition Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Up and Down Arrows</td>
<td>Controls the display order of the columns in the resulting query.</td>
</tr>
<tr>
<td>Column</td>
<td>Displays the column name.</td>
</tr>
<tr>
<td>Alias</td>
<td>Specify an optional column alias. An alias is an alternative column name. Aliases are used to make a column name more descriptive, to shorten the column name, or prevent possible ambiguous references. Note that multibyte characters are not supported in the alias name.</td>
</tr>
</tbody>
</table>
| Condition           | The condition modifies the query’s WHERE clause. When specifying a column condition, you must include the appropriate operator and operand. All standard SQL conditions are supported. For example:  
  
  >=10  
  ='VA'  
  IN (SELECT dept_no FROM dept)  
  BETWEEN SYSDATE AND SYSDATE + 15 |
| Sort Type           | Select ASC (Ascending)  
  or DESC (Descending). |
| Sort Order          | Enter a number (1, 2, 3, and so on) to specify the order in which selected columns should display. |
| Show                | Select this check box to include the column in your query results. You do not need to select Show if you need to add a column to the query for filtering only. For example, suppose you wish to create following query:  
  
  SELECT ename FROM emp WHERE deptno = 10  
  
  To create this query in Query Builder:  
  1. From the Object list, select EMP.  
  2. In the Design Pane, select ename and deptno.  
  3. For the deptno column, in Condition enter =10 and uncheck the Show check box. |
### Condition Attribute Description

<table>
<thead>
<tr>
<th>Condition Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available argument functions include:</td>
<td></td>
</tr>
<tr>
<td>1. <strong>Number columns</strong> - COUNT, COUNT DISTINCT, AVG, MAXIMUM, MINIMUM, SUM</td>
<td></td>
</tr>
<tr>
<td>2. <strong>VARCHAR2, CHAR columns</strong> - COUNT, COUNT DISTINCT, INITCAP, LENGTH, LOWER, LTRIM, RTRIM, TRIM, UPPER</td>
<td></td>
</tr>
<tr>
<td>3. <strong>DATE, TIMESTAMP columns</strong> - COUNT, COUNT DISTINCT</td>
<td></td>
</tr>
<tr>
<td>Specify columns to be used for grouping when an aggregate function is used. Only applicable for columns included in output.</td>
<td></td>
</tr>
<tr>
<td>Deselect the column, excluding it from the query.</td>
<td></td>
</tr>
</tbody>
</table>

As you select columns and define conditions, Query Builder writes the SQL for you. To view the underlying SQL, click the **SQL** tab.

**Creating Relationships Between Objects**

You can create relationships between objects by creating a join. A join identifies a relationship between two or more tables, views, or materialized views.

**About Join Conditions**

When you write a join query, you specify a condition that conveys a relationship between two objects. This condition is called a join condition. A join condition determines how the rows from one object will combine with the rows from another object.

Query Builder supports inner, outer, left, and right joins. An inner join (also called a simple join) returns the rows that satisfy the join condition. An outer join extends the result of a simple join. An outer join returns all rows that satisfy the join condition and returns some or all of those rows from one table for which no rows from the other satisfy the join condition.

**Note:** See *Oracle Database SQL Reference* for information about join conditions.

**Joining Objects Manually**

Create a join manually by selecting the Join column in the Design pane.

1. From the Object Selection pane, select the objects you want to join.

2. Identify the columns you want to join.

You create a join by selecting the Join column adjacent to the column name. The
Join column displays to the right of the datatype. When your cursor is in the appropriate position, the following help tip displays:

3. Select the appropriate Join column for the first object.
When selected, the Join column is darkened. To deselect a Join column, simply select it again or press ESC.

4. Select the appropriate Join column for the second object.
When joined, line connects the two columns. An example is shown in the following figure:

5. Select the columns to be included in your query. You can view the SQL statement resulting from the join by positioning the cursor over the join line.

6. Click **Results** to execute the query.

**Saving a Query**
Once you have built the query and executed it, select the Save button to return to the Report Editor. The query will appear in the SQL Query box.

**Editing a Saved Query**
Once you have saved the query from the Query Builder to the Report Editor, simply select **Query Builder** again to edit the query. The Query Builder will parse the query and present it for modification in the Query Builder interface.
Defining an HTTP Data Set Type

Using the HTTP data source type you can create reports from RSS feeds over the Web.

Note that if you want to include parameters for an HTTP (XML feed), you must define the parameters first, so that they are available for selection when setting up the data source. See Adding Lists of Values and Parameters, page 4-9.

- Enter the URL for the XML feed.
- Select the Method: Get or Post.
- Enter the Username, Password, and Realm for the URL, if required.
- Select the Cache Result box if you wish to cache the results of the query for your session.

By caching the results of the query, multiple templates can be applied to these results without requerying the data. This will enhance online performance. However, if the data is updated during the session, the user cannot view the new data via the View report page until the cache is cleared.

Note: You can control the cache expiration time and the cache size through the configuration settings. See Setting Server Configuration Options, Oracle Business Intelligence Publisher Administrator’s and Developer’s Guide for more information.

- To add a parameter, select the Add link. Enter the Name and select the Value. The Value list is populated by the parameter Identifiers defined in the Parameters section. See Adding Parameters and Lists of Values, page 4-9.
Defining a Web Service Data Set Type

BI Publisher supports document/literal Web service data sources that return the following data types:

- string
- boolean
- dateTime
- decimal
- integer

Tip: If the WSDL URL is outside of your company firewall you must start the BI Publisher server using proxy parameters.

BI Publisher supports Web services that return both simple data types and complex data types. You must make the distinction between simple and complex when you define the Web service data model. See Adding a Simple Web Service, page 5-12 and Adding a Complex Web Service, page 5-16 for descriptions of setting up each type.
Note that if you want to include parameters for the Web service method, you must define the parameters first, so that they are available for selection when setting up the data source. See Adding Parameters and Lists of Values, page 4-9.

Multiple parameters are supported. Ensure the method name is correct and the order of the parameters matches the order in the method. If you want to call a method in your Web service that accepts two parameters, you must map two parameters defined in the report to those two. Note that only parameters of simple type are supported, for example, string and integer.

- Enter the WSDL URL and the Web Service Method.

  Important: Only document/literal Web services are supported.

- To specify a parameter, select the Add link. Select the parameter from the list.

  Note: The parameters must already be set up in the Parameters section of the report definition See Adding Parameters and Lists of Values, page 4-9.

Adding a Simple Web Service Example

This example shows how to add a Web service to BI Publisher as a data source. The Web service returns stock quote information. The Web service will pass one parameter: the quote symbol for a stock.

The WSDL URL is:

http://www.webservicex.net/stockquote.asmx?WSDL

If you are not already familiar with the available methods and parameters in the Web service that you want to call, you can open the URL in a browser to view them. This Web service includes a method called GetQuote. It takes one parameter, which is the stock quote symbol.

To add the Web service as a data source:

1. Enter the Data Set information:
   - Enter a Name for the Data Set and select Web Service as the Type.
   - Select False for Complex Type.
   - Enter the WSDL URL:
     http://www.webservicex.net/stockquote.asmx?WSDL
   - Enter the Method: GetQuote
   - If desired, enter a Time Out period in seconds. If the BI Publisher server cannot
establish a connection to the Web service, the connection attempt will time out after the specified time out period has elapsed.

2. Define the parameter to make it available to the Web service data set.
   Select Parameters on the Report definition pane and click New to create a new parameter. Enter the following:
   - **Identifier** - enter an internal identifier for the parameter.
   - **Data Type** - String
   - **Default Value** - if desired, enter a default for the parameter.
   - **Parameter Type** - Text
   - **Display label** - enter the label you want displayed for your parameter.
   - **Text Field Size** - enter the size for the text entry field in characters.
3. Return to your Web service data set and add the parameter.

   • In the Details section under Parameters, Select **Add**. The Quote parameter you specified is now available from the list.
4. To view the results XML, select **View**. Enter a valid value for your Stock Quote parameter and select **View** again.
Adding a Complex Web Service

You can also add a complex Web service to BI Publisher as a data source. A complex Web service returns complex data types rather than simple string XML.

To use a complex Web service as a data source, select Complex Type equal True, then enter the WSDL URL. After loading and analyzing the WSDL URL, the Report Editor screen will display the available Web services and operations. For each selected operation, the Report Editor will display the structure of the input parameters. By choosing "show optional parameters", you can see all optional parameters as well.

If you are not already familiar with the available methods and parameters in the Web service that you want to call, you can open the URL in a browser to view them.

**To add a complex Web service as a data source:**
1. Enter the Data Set information:
   - Enter a Name for the Data Set and select Web Service as the Type.
   - Select True for Complex Type.
   - Select a security header:
     - Disabled - does not insert a security header.
     - 2002 - enables the "WS-Security" Username Token with the 2002 namespace.
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd

- 2004 - enables the "WS-Security" Username Token with the 2004 namespace:
  http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-usernametoken-profile-1.0#PasswordText

- **Username** and **Password** - enter the username and password for the Web service, if required.

- If desired, enter a **Time Out** period in seconds. If the BI Publisher server cannot establish a connection to the Web service, the connection attempt will time out after the specified time out period has elapsed.

- Enter a **WSDL URL**. When you enter the WSDL, the Web Service list will populate with the available Web services from the WSDL.

- Choose a Web Service from the list. When you choose a Web service from the list, the Method list will populate with the available methods.

- Select the **Method**. When you select the method, the Parameters will display. If you wish to see optional parameters as well, select Show Optional Parameters.

2. Define the parameter to make it available to the Web service data set.
   Select **Parameters** on the **Report** definition pane and click **New** to create a new parameter. Enter the following:
   - **Identifier** - enter an internal identifier for the parameter.
   - **Data Type** - String
   - **Default Value** - if desired, enter a default for the parameter.
   - **Parameter Type** - Text
   - **Display label** - enter the label you want displayed for your parameter.
   - **Text Field Size** - enter the size for the text entry field in characters.

3. Return to your Web service data set and add the parameter.
   - In the entry field for the Parameter, enter the following syntax: 
     `${Parameter.Identifier}` where Parameter.Identifier is the value you entered for Identifier when you defined the parameter to BI Publisher.
4. To view the results XML, select View. Enter a value for your parameter and select View again.

**Defining a Data Template Data Set Type**

Use the BI Publisher data template to create more complex SQL queries. See Building a Data Template, page 6-1 for features and usage. Please note that lexical parameters are only supported when executing a query against an Oracle E-Business Suite instance.

Enter the data template code directly in the **Data Template** text box, or copy and paste the data template from another text source.

**Important:** If copying the data template, the entry in the text box must begin with the `<dataTemplate>` element. Do not include the XML declaration.

---

**Defining an Oracle BI Answers Request Data Set Type**

If you have enabled integration with Oracle Business Intelligence Presentation Services, then you can access the BI catalog to select an Oracle BI Answers request as a data source. Oracle BI Answers is an ad hoc query building tool included in the Oracle
Business Intelligence Enterprise Edition. For more information on building Oracle BI Answers see the *Oracle Business Intelligence Answers, Delivers, and Interactive Dashboards User Guide*.

1. Choose Oracle BI Answers as the data set *Type*.

   **Note:** BI Publisher does not support lists of values and parameters for the Oracle BI Answers request data set type.

2. Select the browse icon to connect to the Oracle BI Answers catalog. This action displays the folders you have access to on the Oracle BI Presentation Services server.

   **Note:** You must set up integration with Oracle BI Presentation Services to enable Oracle BI Answers as a data set Type. See

3. Select the Answers request you wish to use as the data set for your report.

4. Select the *Cache Result* box if you wish to cache the results of the query for your session.

   By caching the results of the query, multiple templates can be applied to these results without requerying the data. This will enhance online performance. However, if the data is updated during the session, the user cannot view the new
data via the View report page until the cache is cleared.

**Note:** You can control the cache expiration time and the cache size through the configuration settings. See Setting Server Configuration Options, *Oracle Business Intelligence Publisher Administrator’s and Developer’s Guide* for more information.

---

**Defining an Oracle BI Discoverer Data Set Type**

For integration with BI Discoverer, you must configure the data source through the Oracle BI Discoverer tab on the Integration page from the Oracle BI Publisher Admin page. See Setting Up Integration with Oracle BI Discoverer, *Oracle Business Intelligence Publisher Administrator’s and Developer’s Guide* for prerequisites, limitations, and setup details.

**Note:** A data model can include only one data set based on a Discoverer Worksheet.

---

**To define the Discoverer Data Set Type:**

1. Select Oracle BI Discoverer from the Data Set Type list. This will enable the appropriate Details region for the Discoverer data source.
2. Enter the Details:
   - **Connection** - click the search icon to launch the list of available Discoverer connections. Navigate to and select the Discoverer connection that owns the worksheet you want to use in your report.

   - **Worksheet** - click the search icon to launch the list of available workbooks. Navigate to and select the workbook and then the worksheet you want to base your report on. When you make your selection, BI Publisher inserts the fully qualified path to the worksheet in the field.

     **Note:** Regardless of whether the Discoverer worksheet is a tabular or crosstab layout, the Discoverer Web service will return the data to Oracle BI Publisher as flat tabular data. This is by design so that you can take full advantage of the layout capabilities in BI Publisher. You can layout this data as a crosstab in BI Publisher.

3. Save your report.

4. Click **View** to ensure that the Discoverer Worksheet you selected returns data to the report. If your worksheet contains parameters, you may need to select values other than the default and click **View** again to enable the worksheet to return data.
Defining a File as a Data Set Type

When you set up data sources (see Setting Up Data Sources, Oracle Business Intelligence Publisher Administrator’s and Developer’s Guide) you can define a file directory as a data source. You can then place xml documents in the file directory to access directly as data sources for your reports.

1. Choose File as the data set Type.

2. Choose the appropriate file directory as the Data Source.

3. Enter the File Name of the XML document to use as the report data set. If the file resides in a subdirectory, include the path.

Defining an MDX Query Data Set Type

BI Publisher supports Multidimensional Expressions (MDX) queries against your OLAP data sources. MDX lets you query multidimensional objects, such as cubes, and return multidimensional cellsets that contain the cube’s data. See your OLAP database documentation for information on the MDX syntax and functions it supports.
1. Select the **Data Source** for this data set. Select the **Default Data Source** (defined in the Report Properties) or select a new data source from the list. Only data sources defined as OLAP connections will display in the list.

2. Select the **Cache Result** box if you wish to cache the results of the query for your session.

By caching the results of the query, multiple templates can be applied to these results without requerying the data. This will enhance online performance. However, if the data is updated during the session, the user cannot view the new data via the View report page until the cache is cleared.

   **Note:** You can control the cache expiration time and the cache size through the configuration settings. See Setting Server Configuration Options, *Oracle Business Intelligence Publisher Administrator’s and Developer’s Guide* for more information.

3. Enter the MDX query by direct entry or by copying and pasting from a third-party MDX editor.

4. Click **Save**.
5. Test your query by selecting the **View** link. This will launch the Report Viewer page. Select **View** again to display the data set returned by your query.

6. Select the **Edit** link to return to the Report Editor.
Building a Data Template

Introduction

The BI Publisher data engine enables you to rapidly generate any kind of XML data structure against any database in a scalable, efficient manner. The data template is the method by which you communicate your request for data to the data engine. It is an XML document whose elements collectively define how the data engine will process the template to generate the XML.

The data engine supports the following functionality:

- Single and multiple data queries
- Query links
- Parameters
- Aggregate functions (SUM, AVG, MIN, MAX, COUNT)
- Event triggers
- Multiple data groups

The XML output generated by the data engine supports the following:

- Unicode for XML Output
  
  Unicode is a global character set that allows multilingual text to be displayed in a single application. This enables you to develop a single multilingual application and deploy it worldwide.

- Canonical format

  The data engine generates date elements using the canonical ISO date format: YYYYY-MM-DDTHH:MI:SS.FF3TZH:TZM for a mapped date element, and
The Data Template Definition

The data template is an XML document that consists of four basic sections: define parameters, define triggers, define data query, define data structure. This structure is shown in the following graphic:

As shown in the sample figure, the data template consists of a `<parameters>` section in which parameters are declared in child `<parameter>` elements; a `<dataQuery>` section in which the SQL queries are defined in child `<sqlStatement>` elements; and a `<dataStructure>` section in which the output XML structure is defined.

The table below lists the elements that make up the XML data template. Each element is described in detail in the following sections. Required elements are noted.
<table>
<thead>
<tr>
<th>Element</th>
<th>Attributes/Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataTemplate (Required)</td>
<td>Attributes:</td>
</tr>
<tr>
<td></td>
<td>• name (Required)</td>
</tr>
<tr>
<td></td>
<td>• description</td>
</tr>
<tr>
<td></td>
<td>• version (Required)</td>
</tr>
<tr>
<td></td>
<td>• defaultPackage - the PL/SQL package name to resolve any lexical references, group filters, or data triggers defined in the template.</td>
</tr>
<tr>
<td></td>
<td>• dataSourceRef - (Required) the default data source reference for the entire data template.</td>
</tr>
<tr>
<td>properties</td>
<td>Consists of one or more <code>&lt;property&gt;</code> elements to support the XML output and Data Engine specific properties.</td>
</tr>
<tr>
<td>property</td>
<td>Attributes:</td>
</tr>
<tr>
<td></td>
<td>• name (Required) - the property name.</td>
</tr>
<tr>
<td></td>
<td>• value - valid values for this property.</td>
</tr>
<tr>
<td>parameters</td>
<td>Consists of one or more <code>&lt;parameter&gt;</code> elements.</td>
</tr>
<tr>
<td>parameter</td>
<td>Attributes:</td>
</tr>
<tr>
<td></td>
<td>• name (Required) - the parameter name that will be referenced in the template.</td>
</tr>
<tr>
<td></td>
<td>• dataType - valid values are: &quot;character&quot;, &quot;date&quot;, &quot;number&quot;</td>
</tr>
<tr>
<td></td>
<td>• defaultValue - value to use for the parameter if none supplied from the data</td>
</tr>
<tr>
<td></td>
<td>• include_in_output - whether this parameter should appear in the XML output or not. The valid values are &quot;true&quot; and &quot;false&quot;.</td>
</tr>
<tr>
<td>lexicals</td>
<td>(Supported for queries against the Oracle E-Business Suite only). Consists of one or more lexical elements to support flexfields.</td>
</tr>
<tr>
<td>Element</td>
<td>Attributes/Description</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------</td>
</tr>
</tbody>
</table>
| lexical     | There are four types of key flexfield-related lexicals as follows:  
  • oracle.apps.fnd.flex.kff.segments_metadata  
  • oracle.apps.fnd.flex.kff.select  
  • oracle.apps.fnd.flex.kff.where  
  • oracle.apps.fnd.flex.kff.order_by |
| dataQuery   | Consists of one or more `<sqlstatement>` or `<xml>` elements. |
| sqlstatement| Attributes:  
  • name (Required) - the unique query identifier. Note that this name identifier will be the same across the data template. Enter the query inside the CDATA section. |
| xml         | Attributes:  
  • name (Required) - the unique query identifier.  
  • expressionPath – Xpath expression |
| url         | Attributes:  
  • method – either GET or POST  
  • realm - authentication name  
  • username - valid username  
  • password - valid password |
<table>
<thead>
<tr>
<th>Element</th>
<th>Attributes/Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>link</td>
<td>Attributes:</td>
</tr>
<tr>
<td></td>
<td>• parentQuery - specify the parent query name.</td>
</tr>
<tr>
<td></td>
<td>• parentColumn - specify the parent column name.</td>
</tr>
<tr>
<td></td>
<td>• childQuery - specify the child query name.</td>
</tr>
<tr>
<td></td>
<td>• childColumn - specify the child column name.</td>
</tr>
<tr>
<td></td>
<td>• condition - the SQL operator that defines the relationship between the parent column and the child column. The following values for condition are supported: (=, &lt;, \leq, &gt;, \geq).</td>
</tr>
<tr>
<td>dataTrigger</td>
<td>Attributes:</td>
</tr>
<tr>
<td></td>
<td>• name (Required) - the event name to fire this trigger</td>
</tr>
<tr>
<td></td>
<td>• source (Required) - the PL/SQL (&lt;package name&gt;.&lt;function name&gt;)</td>
</tr>
<tr>
<td>dataStructure</td>
<td>(Required for multiple queries) Defines the structure of the output XML. Consists of (&lt;group&gt;) and (&lt;element&gt;) elements to specify the structure. This section is optional for single queries; if not specified, the data engine will generate flat XML.</td>
</tr>
<tr>
<td>group</td>
<td>Consists of one or more (&lt;element&gt;) elements and sub (&lt;group&gt;) elements. Attributes:</td>
</tr>
<tr>
<td></td>
<td>• name (Required) - the XML tag name to be assigned to the group.</td>
</tr>
<tr>
<td></td>
<td>• source (Required) - the unique query identifier for the corresponding sqlstatement from which the group’s elements will be derived.</td>
</tr>
<tr>
<td></td>
<td>• groupFilter - the filter to apply to the output data group set. Define the filter as: (&lt;package name&gt;.&lt;function name&gt;).</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> Applying a filter has performance impact. Do not use this functionality unless necessary. When possible, filter data using a WHERE clause in your query.</td>
</tr>
</tbody>
</table>
### Constructing the Data Template

You can use any text or XML editor to write a data template.

### Data Template Declaration

The `<dataTemplate>` element is the root element. It has a set of related attributes expressed within the `<dataTemplate>` tag.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>(Required) Enter the data template name.</td>
</tr>
<tr>
<td>description</td>
<td>(Optional) Enter a description of this data template.</td>
</tr>
<tr>
<td>version</td>
<td>(Required) Enter a version number for this data template.</td>
</tr>
<tr>
<td>defaultPackage</td>
<td>This attribute is required if your data template contains lexical references or any other calls to PL/SQL.</td>
</tr>
<tr>
<td>dataSourceRef</td>
<td>(Required) The default data source reference for the entire data template.</td>
</tr>
</tbody>
</table>

### Properties Section

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

Example:
The following table shows the supported properties:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>include_parameters</td>
<td>Indicates whether to include parameters in the output. Valid values are:</td>
</tr>
<tr>
<td></td>
<td>• True (default)</td>
</tr>
<tr>
<td></td>
<td>• False</td>
</tr>
<tr>
<td>include_null_Element</td>
<td>Indicates whether to remove or keep the null elements in the output. Valid values are:</td>
</tr>
<tr>
<td></td>
<td>• True (default)</td>
</tr>
<tr>
<td></td>
<td>• False</td>
</tr>
<tr>
<td>xml_tag_case</td>
<td>Allows you to set the case for the output XML element names. Valid values are:</td>
</tr>
<tr>
<td></td>
<td>• upper (default)</td>
</tr>
<tr>
<td></td>
<td>• lower</td>
</tr>
<tr>
<td></td>
<td>• as_are (The case will follow the definition in the dataStructure section.)</td>
</tr>
<tr>
<td>db_fetch_size</td>
<td>Sets the number of rows fetched at a time through the jdbc connection. The default value is 500.</td>
</tr>
</tbody>
</table>

<properties>
  <property name="include_parameters" value="false" />
  <property name="include_null_Element" value="false" />
  <property name="include_rowsettag" value="false" />
  <property name="scalable_mode" value="on" />
</properties>
<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scalable_mode</td>
<td>Sets the data engine to execute in scalable mode. This is required when processing a large volume of data.</td>
</tr>
<tr>
<td></td>
<td>Valid values:</td>
</tr>
<tr>
<td></td>
<td>• on</td>
</tr>
<tr>
<td></td>
<td>• off (default)</td>
</tr>
<tr>
<td>include_rowsettag</td>
<td>Allows you to include or exclude the Rowset Tag from the output.</td>
</tr>
<tr>
<td></td>
<td>Valid values:</td>
</tr>
<tr>
<td></td>
<td>• true (default)</td>
</tr>
<tr>
<td></td>
<td>• false</td>
</tr>
<tr>
<td>debug_mode</td>
<td>Turns debug mode on or off.</td>
</tr>
<tr>
<td></td>
<td>Valid values:</td>
</tr>
<tr>
<td></td>
<td>• on</td>
</tr>
<tr>
<td></td>
<td>• off (default)</td>
</tr>
</tbody>
</table>

**Parameters Section**

A parameter is a variable whose value can be set at runtime. Parameters are especially useful for modifying SELECT statements and setting PL/SQL variables at runtime. The Parameters section of the data template is optional.

**How to Define Parameters**

The `<parameter>` element is placed between the open and close `<parameters>` tags. The `<parameter>` element has a set of related attributes. These are expressed within the `<parameter>` tag. For example, the `name`, `dataType`, and `defaultValue` attributes are expressed as follows:

```xml
<parameters>
  <parameter name="dept" dataType="number" defaultValue="10"/>
</parameters>
```
### Attribute Name Description

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><strong>Required.</strong> A keyword, unique within a given Data Template, that identifies the parameter.</td>
</tr>
<tr>
<td>dataType</td>
<td>Optional. Specify the parameter data type as &quot;character&quot;, &quot;date&quot;, or &quot;number&quot;. Default value is &quot;character&quot;. For the &quot;date&quot; dataType, the following three formats (based on the canonical ISO date format) are supported:</td>
</tr>
<tr>
<td></td>
<td>• YYYY-MM-DD (example: 1997-10-24)</td>
</tr>
<tr>
<td></td>
<td>• YYYY-MM-DD HH24:MI:SS (example: 1997-10-24 12:00:00)</td>
</tr>
<tr>
<td></td>
<td>• YYYY-MM-DDTHH24:MI:SS.FF3TZH:TZM</td>
</tr>
<tr>
<td>defaultValue</td>
<td>Optional. This value will be used for the parameter if no other value is supplied from the data at runtime.</td>
</tr>
<tr>
<td>include_in_output</td>
<td>Optional. Whether this parameter should appear in XML output or not. The valid values are &quot;true&quot; and &quot;false&quot;.</td>
</tr>
</tbody>
</table>

### How to Pass Parameters

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

**Query:**

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

*At runtime, the value of department is passed to the query:*  
```sql
SELECT * FROM EMP  
WHERE deptno=10
```

### Data Query Section

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

### Supported Column Types

The following column types are selectable:

• VARCHAR2, CHAR
• NUMBER
• DATE, TIMESTAMP
• BLOB/BFILE (conditionally supported)
  BLOB image retrieval is supported in the following two cases:
  • Using the SetSQL API (see SQL to XML Processor, page 6-28)
  • In the data template when no Structure section is defined. The returned data
    must be flat XML.

The BLOB/BFILE must be an image. Images are retrieved into your results XML as
base64 encoding. You can retrieve any image type that is supported in the RTF
template (jpg, gif, or png). You must use specific syntax to render the retrieved
image in your template. See Rendering an Image Retrieved from BLOB Data, page
7-17.

• CLOB (conditionally supported)
  The CLOB must contain text or XML. Data cannot be escaped inside the CLOB
column.

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

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

How to Define SQL Queries
The <sqlStatement> element is placed between the open and close dataQuery tags.
The <sqlStatement> element has a related attribute, name. It is expressed within the
<sqlStatement> tag. The query is entered in the CDATA section. For example:

```xml
<dataQuery>
  <sqlStatement name="Q1">
    <![CDATA[SELECT DEPTNO, DNAME, LOC from dept]]>
  </sqlStatement>
</dataQuery>
```
<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>A unique identifying name for the query. Note that this name will be referred to throughout the data template.</td>
</tr>
</tbody>
</table>

If your column names are not unique, you must use aliases in your SELECT statements to ensure the uniqueness of your column names. If you do not use an alias, then the default column name is used. This becomes important when you specify the XML output in the dataStructure section. To specify an output XML element from your query you declare a `value` attribute for the element tag that corresponds to the source column.

**Tip:** Performing operations in SQL is faster than performing them in the data template or PL/SQL. It is recommended that you use SQL for the following operations:

- Use a WHERE clause instead of a group filter to exclude records.
- Perform calculations directly in your query rather than in the template.

**Lexical References**

You can use lexical references to replace the clauses appearing after SELECT, FROM, WHERE, GROUP BY, ORDER BY, or HAVING. Use a lexical reference when you want the parameter to replace multiple values at runtime.

Create a lexical reference using the following syntax:

```
&parametername
```

Define the lexical parameters as follows:

- Before creating your query, define a parameter in the PL/SQL default package for each lexical reference in the query. The data engine uses these values to replace the lexical parameters.
- Create your query containing lexical references.

For example:
How to Define a Data Link Between Queries

If you have multiple queries, you must link them to create the appropriate data output. In the data template, there are two methods for linking queries: using bind variables or using the `<link>` element to define the link between queries.

**Tip:** To maximize performance when building data queries in the data template:

- BI Publisher tests have shown that using bind variables is more efficient than using the link tag.

The following example shows a query link using a bind variable:

```xml
<dataQuery>
  <sqlstatement name="Q1">
    <![CDATA[SELECT EMPNO, ENAME, JOB from EMP
      WHERE DEPTNO = :DEPTNO]]>
  </sqlstatement>
</dataQuery>
```

The `<link>` element has a set of attributes. Use these attributes to specify the required link information. You can specify any number of links. For example:

```xml
<link name="DEPTEMP_LINK" parentQuery="Q1" parentColumn="DEPTNO"
      childQuery="Q_2" childColumn="DEPARTMENTNO"/>
```

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required. Enter a unique name for the link.</td>
</tr>
</tbody>
</table>
### Attribute Name | Description
--- | ---
parentQuery | Specify the parent query name. This must be the name that you assigned to the corresponding `<sqlstatement>` element. See How to Define Queries, page 6-10.
parentColumn | Specify the parent column name.
childQuery | Specify the child query name. This must be the name that you assigned to the corresponding `<sqlstatement>` element. See How to Define Queries, page 6-10.
childColumn | Specify the child column name.

### Using Data Triggers

Data triggers execute PL/SQL functions at specific times during the execution and generation of XML output. Using the conditional processing capabilities of PL/SQL for these triggers, you can do things such as perform initialization tasks and access the database.

Data triggers are optional, and you can have as many `<dataTrigger>` elements as necessary.

The `<dataTrigger>` element has a set of related attributes. These are expressed within the `<dataTrigger>` tag. For example, the `name` and `source` attributes are expressed as follows:

```xml
<dataTrigger name="beforeReport" source="employee.beforeReport()"/>
<dataTrigger name="beforeReport" source="employee.beforeReport(:Parameter)"/>
```

### Attribute Name | Description
--- | ---
name | The event name to fire this trigger.
source | The PL/SQL `<package name>;<function name>` where the executable code resides.

The location of the trigger indicate at what point the trigger fires:

- Place a beforeReport trigger anywhere in your data template before the `<dataStructure>` section. A beforeReport trigger fires before the dataQuery is executed.
• Place an afterReport trigger after the <dataStructure> section. An afterReport trigger fires after you exit and after XML output has been generated.

Data Structure Section

In the data structure section you define what the XML output will be and how it will be structured. The complete group hierarchy is available for output. You can specify all the columns within each group and break the order of those columns; you can use summaries, and placeholders to further customize within the groups. The dataStructure section is required for multiple queries and optional for single queries. If omitted for a single query, the data engine will generate flat XML.

Defining a Group Hierarchy

In the data template, the <group>element is placed between open and close <dataStructure> tags. Each <group> has a set of related elements. You can define a group hierarchy and name the element tags for the XML output.

Creating Break Groups

Use a break group to produce subtotals or add placeholder columns. A break group suppresses duplicate values in sequential records. You should set an Order By clause in the SQL query to suppress duplicate values.

Assign a name to the group, and declare the source query, then specify the elements you want included in that group. When you specify the element, you assign it a name that will be used as the XML output tag name, and you declare the source column as the value. If you do not assign a name, the value (or source column name) will be used as the tag name.

For example:

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

The following table lists the attributes for the <group> element tag:
Attribute Name | Description
---|---
**name** | Specify any unique name for the group. This name will be used as the output XML tag name for the group.

**source** | The name of the query that provides the source data for the group. The source must come from the name attribute of the `<sqlStatement>` element.

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

| Attribute Name | Description |
---|---|
**name** | Specify any name for the element. This name will be used as the output XML tag name for the element. The name is optional. If you do not specify a name, the source column name will be used as the XML tag name.

**value** | The name of the column that provides the source data for the element (from your query).

**Applying Group Filters**

It is strongly recommended that you use a WHERE clause instead of a group filter to exclude records from your extract. Filters enable you to conditionally remove records selected by your queries, however, this approach impacts performance. Groups can have user-created filters, using PL/SQL.

The PL/SQL function must return a boolean value (TRUE or FALSE). Depending on whether the function returns TRUE or FALSE, the current record is included or excluded from the XML data output.

For example, a sample PL/SQL function might be:

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

An example of the group filter in your data template definition would be:
Creating a Summary Column

A summary column performs a computation on another column’s data. Using the `function` attribute of the `<element>` tag, you can create the following summaries: sum, average, count, minimum, and maximum.

To create a summary column, you must define the following three attributes in the `<element>` tag:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The XML tag name to be used in the XML data output.</td>
</tr>
<tr>
<td>source</td>
<td>The name of the column that contains the data on which the summary calculation is to be performed. The source column remains unchanged.</td>
</tr>
<tr>
<td>function</td>
<td>The aggregation function to be performed. The type tells the XDO data engine how to compute the summary column values. Valid values are: SUM(), AVG(), COUNT(), MAX(), and MIN().</td>
</tr>
</tbody>
</table>

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

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

Flexfield Support

**Note:** This section applies to data templates written to query the Oracle Applications database.
Flexfields are defined in the data template using lexical parameters.

**How to define a flexfield**

1. Define the SELECT statement to use for the report data.

2. Within the SELECT statement, define each flexfield as a lexical. Use the &LEXICAL_TAG to embed flexfield related lexicals into the SELECT statement.

3. Define the flexfield-related lexicals using XML tags in the data template.

**Example**

```xml
<dataTemplate ...>
  <parameters ... />
  <lexicals ...>
    <lexical type="oracle.apps.fnd.flex.kff.segments_metadata" name="<Name of the lexical>" comment="<comment>" />
    <lexical type="oracle.apps.fnd.flex.kff.select" name="<Name of the lexical>" comment="<comment>" />
  </lexicals>
  <dataQuery>
    <sqlStatement ...>
      SELECT &FLEX_SELECT flex_select_alias
      FROM some_table st, code_combination_table cct
      WHERE st.some_column = 'some_condition'
      AND &FLEX_WHERE
      ORDER BY st.some_column, &FLEX_ORDER_BY
    </sqlStatement>
  </dataQuery>
</dataTemplate>
```

**Flexfield Lexicals**

There are four types of KFF-related lexicals. These are:

- oracle.apps.fnd.flex.kff.segments_metadata
- oracle.apps.fnd.flex.select
- oracle.apps.fnd.flex.kff.where
- oracle.apps.fnd.flex.kff.order_by

Following are descriptions of each type of KFF lexical:
oracle.apps.fnd.flex.kff.segments_metadata

Use this type of lexical to retrieve flexfield-related metadata. Using this lexical, you are not required to write PL/SQL code to retrieve this metadata. Instead, define a dummy SELECT statement, then use this lexical to get the metadata.

The XML syntax for this lexical is as follows:

```xml
<lexicals>
  <lexical
    type="oracle.apps.fnd.flex.kff.segments_metadata"
    name="Name of the lexical"
    comment="Comment"
    application_short_name="Application Short Name of the KFF"
    id_flex_code="Internal code of the KFF"
    id_flex_num="Internal number of the KFF structure"
    segments="For which segment(s) is this metadata requested?"
    show_parent_segments="Should the parent segments be listed?"
    metadata_type="Type of metadata requested"/>
</lexicals>
```

The following table lists the attributes for the segments_metadata lexical:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>application_short_name</td>
<td>(Required) The application short name of the key flexfield. For example: SQLGL.</td>
</tr>
<tr>
<td>id_flex_code</td>
<td>(Required) the internal code of the key flexfield. For example: GL#</td>
</tr>
<tr>
<td>id_flex_num</td>
<td>(Required) Internal number of the key flexfield structure. For example: 101</td>
</tr>
<tr>
<td>segments</td>
<td>(Optional) Identifies for which segments this data is requested. Default value is &quot;ALL&quot;. See the Oracle Applications Developer’s Guide for syntax.</td>
</tr>
<tr>
<td>show_parent_segments</td>
<td>(Optional) Valid values are &quot;Y&quot; and &quot;N&quot;. Default value is &quot;Y&quot;. If a dependent segment is displayed, the parent segment is automatically displayed, even if it is not specified as displayed in the segments attribute.</td>
</tr>
<tr>
<td>metadata_type</td>
<td>(Required) Identifies what type of metadata is requested. Valid values are:</td>
</tr>
<tr>
<td></td>
<td>above_prompt - above prompt of segment(s).</td>
</tr>
<tr>
<td></td>
<td>left_prompt - left prompt of segment(s)</td>
</tr>
</tbody>
</table>
Example
This example shows how to request the `above_prompt` of the GL Balancing Segment, and the `left_prompt` of the GL Account Segment.

```sql
SELECT &FLEX_GL_BALANCING_APROMPT alias_gl_balancing_aprompt,
&FLEX_GL_ACCOUNT_LPROMPT alias_gl_account_lprompt
FROM dual
```

```xml
<lexicals>
<lexical type="oracle.apps.fnd.flex.kff.segments_metadata"
name="FLEX_GL_BALANCING_APROMPT"
comment="Comment"
application_short_name="SQLGL"
id_flex_code="GL#"
id_flex_num=":P_ID_FLEX_NM"
segments="GL_BALANCING"
metadata_type="ABOVE_PROMPT"/>
<lexical type="oracle.apps.fnd.flex.kff.segments_metadata"
name="FLEX_GL_ACCOUNT_LPROMPT"
comment="Comment"
application_short_name="SQLGL"
id_flex_code="GL#"
id_flex_num=":P_ID_FLEX_NUM"
segments="GL_ACCOUNT"
metadata_type="LEFT_PROMPT"/>
</lexicals>
```

**oracle.apps.fnd.flex.kff.select**

This type of lexical is used in the SELECT section of the statement. It is used to retrieve and process key flexfield (kff) code combination related data based on the lexical definition.

The syntax is as follows:

```xml
<lexicals>
<lexical type="oracle.apps.fnd.flex.kff.select"
name="Name of the lexical"
comment="Comment"
application_short_name="Application Short Name of the KFF"
id_flex_code="Internal code of the KFF"
id_flex_num="Internal number of the KFF structure"
multiple_id_flex_num="Are multiple structures allowed?"
code_combination_table_alias="Code Combination Table Alias"
segments="Segments for which this data is requested"
show_parent_segments="Should the parent segments be listed?"
output_type="output type"/>
</lexicals>
```

The following table lists the attributes for this lexical:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>application_short_name</td>
<td>(Required) The application short name of the key flexfield. For example: SQLGL.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>id_flex_code</td>
<td>(Required) the internal code of the key flexfield. For example: GL#</td>
</tr>
<tr>
<td>id_flex_num</td>
<td>(Conditionally required) Internal number of the key flexfield structure. For example: 101. Required if MULTIPLE_ID_FLEX_NUM is &quot;N&quot;.</td>
</tr>
<tr>
<td>multiple_id_flex_num</td>
<td>(Optional) Indicates whether this lexical supports multiple structures or not. Valid values are &quot;Y&quot; and &quot;N&quot;. Default is &quot;N&quot;. If set to &quot;Y&quot;, then flex will assume all structures are potentially used for data reporting and it will use <code>&lt;code_combination_table_alias&gt;.&lt;setting_column_name&gt;</code> to retrieve the structure number.</td>
</tr>
<tr>
<td>code_combination_table_alias</td>
<td>(Optional) Segment column names will be prepended with this alias.</td>
</tr>
<tr>
<td>segments</td>
<td>(Optional) Identifies for which segments this data is requested. Default value is &quot;ALL&quot;. See the Oracle Applications Developer’s Guide for syntax.</td>
</tr>
<tr>
<td>show_parent_segments</td>
<td>(Optional) Valid values are &quot;Y&quot; and &quot;N&quot;. Default value is &quot;Y&quot;. If a dependent segment is displayed, the parent segment is automatically displayed, even if it is not specified as displayed in the segments attribute.</td>
</tr>
<tr>
<td>output_type</td>
<td>(Required) Indicates what kind of output should be used as the reported value. Valid values are: value - segment value as it is displayed to user. padded_value - padded segment value as it is displayed to user. Number type values are padded from the left. String type values are padded on the right.</td>
</tr>
<tr>
<td>description</td>
<td>Segment value’s description up to the description size defined in the segment definition.</td>
</tr>
<tr>
<td>full_description</td>
<td>Segment value’s description (full size).</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>security</td>
<td>Returns Y if the current combination is secured against the current user, N otherwise.</td>
</tr>
</tbody>
</table>

**Example**

This example shows how to report concatenated values, concatenated descriptions, the value of the GL Balancing Segment, and the full description of the GL Balancing Segment for a single structure:
SELECT &FLEX_VALUE_ALL alias_value_all,
&FLEX_DESCR_ALL alias_descr_all,
&FLEX_GL_BALANCING alias_gl_balancing,
&FLEX_GL_BALANCING_FULL_DESCR alias_gl_balancing_full_descr,
...
FROM gl_code_combinations gcc,
some_other_gl_table sogt
WHERE gcc.chart_of_accounts_id = :p_id_flex_num
and sogt.code_combination_id = gcc.code_combination_id
and <more conditions on sogt>
This type of lexical is used in the WHERE section of the statement. It is used to modify the WHERE clause such that the SELECT statement can filter based on key flexfield segment data.

The syntax for this lexical is as follows:

```xml
<lexicals>
  <lexical
    type="oracle.apps.fnd.flex.kff.where"
    name="Name of the lexical"
    comment="Comment"
    application_short_name="Application Short Name of the KFF"
    id_flex_code="Internal code of the KFF"
    id_flex_num="Internal number of the KFF structure"
    code_combination_table_alias="Code Combination Table Alias"
    segments="Segments for which this data is requested"
    operator="The boolean operator to be used in the condition"
    operand1="Values to be used on the right side of the operator"
    operand2="High value for the BETWEEN operator"/>
</lexicals>
```

The attributes for this lexical are listed in the following table:

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

operand2 (Optional) High value for the BETWEEN operator.

full_description Segment value's description (full size).

security Returns Y if the current combination is secured against the current user, N otherwise.

Example
This example shows a filter based on the GL Account segment and the GL Balancing Segment:

```sql
SELECT <some columns>
FROM gl_code_combinations gcc, some_other_gl_table sogt
WHERE gcc.chart_of_accounts_id = :p_id_flex_num
and sogt.code_combination_id = gcc.code_combination_id
and &FLEX_WHERE_GL_ACCOUNT
and &FLEX_WHERE_GL_BALANCING
and <more conditions on sogt>

<lexicals>
<lexical
 type="oracle.apps.fnd.flex.kff.where"
 name="FLEX_WHERE_GL_ACCOUNT"
 comment="Comment"
 application_short_name="SQLGL"
 id_flex_code="GL#"
 id_flex_num=":P_ID_FLEX_NUM"
 code_combination_table_alias="gcc"
 segments="GL_ACCOUNT"
 operator="=
 operand1=":P_GL_ACCOUNT"/>

<lexical
 type="oracle.apps.fnd.flex.kff.where"
 name="FLEX_WHERE_GL_BALANCING"
 comment="Comment"
 application_short_name="SQLGL"
 id_flex_code="GL#"
 id_flex_num=":P_ID_FLEX_NUM"
 code_combination_table_alias="gcc"
 segments="GL_BALANCING"
 operator="BETWEEN"
 operand1=":P_GL_BALANCING_LOW"
 operand2=":P_GL_BALANCING_HIGH"/>
</lexicals>

oracle.apps.fnd.flex.kff.order_by
This type of lexical is used in the ORDER BY section of the statement. It returns a list of column expressions so that the resulting output can be sorted by the flex segment values.

The syntax for this lexical is as follows:
The attributes for this lexical are listed in the following table:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>application_short_name</td>
<td>(Required) The application short name of the key flexfield. For example: SQLGL.</td>
</tr>
<tr>
<td>id_flex_code</td>
<td>(Required) the internal code of the key flexfield. For example: GL#</td>
</tr>
<tr>
<td>id_flex_num</td>
<td>(Conditionally required) Internal number of the key flexfield structure. For example: 101. Required if MULTIPLE_ID_FLEX_NUM is &quot;N&quot;.</td>
</tr>
<tr>
<td>multiple_id_flex_num</td>
<td>(Optional) Indicates whether this lexical supports multiple structures or not. Valid values are &quot;Y&quot; and &quot;N&quot;. Default is &quot;N&quot;. If set to &quot;Y&quot;, then flex will assume all structures are potentially used for data reporting and it will use &lt;code_combination_table_alias&gt;.&lt;set_defining_column_name&gt; to retrieve the structure number.</td>
</tr>
<tr>
<td>code_combination_table_alias</td>
<td>(Optional) Segment column names will be prepended with this alias.</td>
</tr>
<tr>
<td>segments</td>
<td>(Optional) Identifies for which segments this data is requested. Default value is &quot;ALL&quot;. See the Oracle Applications Developer’s Guide for syntax.</td>
</tr>
<tr>
<td>show_parent_segments</td>
<td>(Optional) Valid values are &quot;Y&quot; and &quot;N&quot;. Default value is &quot;Y&quot;. If a dependent segment is displayed, the parent segment is automatically displayed, even if it is not specified as displayed in the segments attribute.</td>
</tr>
</tbody>
</table>
Example
The following example shows results sorted based on GL Account segment and GL Balancing segment for a single structure KFF.

```sql
SELECT <some columns>
FROM gl_code_combinations gcc,
     some_other_gl_table sogt
WHERE gcc.chart_of_accounts_id = :p_id_flex_num
  AND sogt.code_combination_id = gcc.code_combination_id
  AND <more conditions on sogt>
ORDER BY <some order by columns>,
    &FLEX_ORDER_BY_GL_ACCOUNT,
    &FLEX_ORDER_BY_GL_BALANCING
```

<lexicals>
  <lexical
    type="oracle.apps.fnd.flex.kff.order_by"
    name="FLEX_ORDER_BY_GL_ACCOUNT"
    comment="Comment"
    application_short_name="SQLGL"
    id_flex_code="GL#"
    id_flex_num=":P_ID_FLEX_NUM"
    code_combination_table_alias="gcc"
    segments="GL_ACCOUNT"
    show_parent_segments="Y"/>
  <lexical
    type="oracle.apps.fnd.flex.kff.order_by"
    name="FLEX_ORDER_BY_GL_BALANCING"
    comment="Comment"
    application_short_name="SQLGL"
    id_flex_code="GL#"
    id_flex_num=":P_ID_FLEX_NUM"
    code_combination_table_alias="gcc"
    segments="GL_BALANCING"
    show_parent_segments="N"/>
</lexicals>

Using the Data Engine Java API
This section describes how to utilize BI Publisher's data engine outside of the BI Publisher Enterprise user interface through the Java APIs. Use the descriptions in this section in conjunction with the Javadocs included with your installation files.

Calling a Data Template from the Java API

The following classes comprise the data engine utility Java API:

- oracle.apps.xdo.oa.util.DataTemplate (OA wrapper API)
- oracle.apps.xdo.dataengine.DataProcessor (Core wrapper API)

The DataProcessor class is the main class to use to execute a data template with the BI Publisher Data Engine. To use this API, you will need to instantiate this class and set parameter values for the data template, connection and output destination. Once the parameters are set, you can start processing by calling processData() method.
Example
This example provides a sample data template file, then shows an annotated Java code sample of how to call it.

The sample data template is called EmpDataTemplate.xml and is stored as /home/EmpDataTemplate.xml:

```xml
<?xml version="1.0" encoding="WINDOWS-1252" ?>
<dataTemplate name="EmpData" description="Employee Details" Version="1.0">
  <parameters>
    <parameter name="p_DeptNo" dataType="character" />
  </parameters>
  <dataQuery>
    <sqlStatement name="Q1">
      <![CDATA[
      SELECT d.DEPTNO,d.DNAME,d.LOC,EMPNO,ENAME,JOB,MGR,HIREDATE,
      SAL,nvl(COMM,0)
      FROM dept d, emp e
      WHERE d.deptno=e.deptno
      AND d.deptno = nvl(:p_DeptNo,d.deptno)
      ]]></sqlStatement>
  </dataQuery>
  <dataStructure>
    <group name="G_DEPT" source="Q1">
      <element name="DEPT_NUMBER" value="DEPTNO" />
      <element name="DEPT_NAME" value="DNAME" />
      <element name="DEPTSAL" value="G_EMP.SALARY" function="SUM()" />
      <element name="LOCATION" value="LOC" />
    </group>
    <group name="G_EMP" source="Q1">
      <element name="EMPLOYEE_NUMBER" value="EMPNO" />
      <element name="NAME" value="ENAME" />
      <element name="JOB" value="JOB" />
      <element name="MANAGER" value="MGR" />
      <element name="HIREDATE" value="HIREDATE" />
      <element name="SALARY" value="SAL" />
    </group>
  </dataStructure>
</dataTemplate>
```

The following code sample is an annotated snippet of the Java code used to process the data template by the data engine:
try {

//Initialization – instantiate the DataProcessor class/
DataProcessor dataProcessor = new DataProcessor();

//Set Data Template to be executed
dataProcessor.setDataTemplate("/home/EmpDataTemplate.xml");

//Get Parameters – this method will return an array of the
//parameters in the data template
ArrayList parameters = dataProcessor.getParameters();

// Now we have the arraylist we need to iterate over
// the parameters and assign values to them
Iterator it = parameters.iterator();

while (it.hasNext())
{
    Parameter p = (Parameter) it.next();
    if (p.getName().equals("p_DeptNo"))
    // Here we assign the value '10' to the p_DeptNo parameter.
    // This could have been entered from a report submission
    // screen or passed in from another process.
    p.setValue("10");
}

// The parameter values now need to be assigned
// to the data template; there are two methods
// available to do this: 1. Use the setParameters
// method to assign the 'parameters' object to the template:
dataProcessor.setParameters(parameters);

// 2. or you can assign parameter values using a hashtable.

Hashtable parameters = new Hashtable();
parameters.put("p_DeptNo", "10");
dataProcessor.setParameters(parameters);

// Now set the jdbc connection to the database that you
// wish to execute the template against.
// This sample assumes you have already created
// the connection object 'jdbcConnection'
dataProcessor.setConnection(jdbcConnection);

// Specify the output directory and file for the data file
dataProcessor.setOutput("/home/EmpDetails.xml");

// Process the data template
dataProcessor processData();
}
catch (Exception e)
{
}
}

**SQL to XML Processor**

The data engine not only supports data generation from data templates, but it can also return data by simply passing it a SQL statement. This functionality is similar to the native database support for generating XML with the added advantage that you can retrieve huge amounts of data in a hierarchical format without sacrificing performance and memory consumption. You SQL statement can also contain parameters that can be given values prior to final processing.

The processor will generate XML in a ROWSET/ROW format. The tag names can be overridden using the setRowsetTag and setRowsTag methods.
The following annotated code sample shows how to use the `setSQL` method to pass a SQL statement to the data engine and set the element names for the generated data:

**Example**

```java
// Initialization - instantiate the DataProcessor class
DataProcessor dataProcessor = new DataProcessor();
// Set the SQL to be executed
dataProcessor.setSQL("select invoicenum, invoiceval
from invoice_table where supplierid = :SupplID");
// Setup the SuppID value to be used
Hashtable parameters = new Hashtable();
parameters.put("SupplID ", "2000");
// Set the parameters
dataProcessor.setParameters(parameters);
// Set the db connection
dataProcessor.setConnection(jdbcConnection);
// Specify the output file name and location
dataProcessor.setOutput("/home/InvoiceDetails.xml");
// Specify the root element tag name for the generated output
dataProcessor.setRowsetTag("INVOICES");
// Specify the row elemen tag name for the generated output
dataProcessor.setRowsetTag("INVOICE");
// Execute the SQL
dataProcessor.processData();
```

**Other Useful Methods**

The data engine has several very useful functions that can be used to generate objects or files that can be used with the other BI Publisher APIs:

- **writeDefaultLayout** – once the DataTemplate has been instantiated you can call this method to generate a default RTF template that can be used with the RTFProcessor to create an XSL template to be used with the FOPProcessor. Alternatively, the default RTF can be loaded into Microsoft Word for further formatting. This method can generate either a String or Stream output.

- **writeXMLSchema** - once the DataTemplate has been instantiated you can call this method to generate an XML schema representation of your data template. This is very useful if you are working with PDF templates and need to create mapping from the PDF document to your XML data.

- **setScalableModeOn** – if you know you are going to return a large dataset or have a long running query you can specify that the data engine enter scalable mode. This will cause it to use the disk rather than use memory to generate the output.

- **setMaxRows** – this allows you to specify a fixed number of rows to be returned by the engine. This is especially useful when you want to generate some sample data to build a layout template against.

**Sample Data Templates**

This section contains two sample data templates:
• Employee Listing

• General Ledger Journals Listing

The sample files are annotated to provide a better description of the components of the data template. To see more data template samples, see the BI Publisher page on Oracle Technology Network (OTN) [http://www.oracle.com/technology/products/applications/publishing/index.html]. From here you can copy and paste the samples to get you started on your own data templates.

**Employee Listing Data Template**

This template extracts employee data and department details. It has a single parameter, Department Number, that has to be populated at runtime. The data is extracted using two joined queries that use the bind variable method to join the parent (Q1) query with the child (Q2) query. It also uses the event trigger functionality using a PL/SQL package "employee" to set the where clause on the Q1 query and to provide a group filter on the G_DEPT group.

The sample data template will generate the following XML:
<?xml version="1.0" encoding="UTF-8"?>
<dataTemplateName>
  <LIST_G_DEPT>
    <G_DEPT>
      <DEPT_NUMBER>10</DEPT_NUMBER>
      <DEPT_NAME>ACCOUNTING</DEPT_NAME>
      <LOCATION>NEW YORK</LOCATION>
    <LIST_G_EMP>
      <G_EMP>
        <EMPLOYEE_NUMBER>7782</EMPLOYEE_NUMBER>
        <NAME>CLARK</NAME>
        <JOB>MANAGER</JOB>
        <MANAGER>7839</MANAGER>
        <HIREDATE>1981-06-09T00:00:00.000-07:00</HIREDATE>
        <SALARY>2450</SALARY>
    <G_EMP>
      <EMPLOYEE_NUMBER>7839</EMPLOYEE_NUMBER>
      <NAME>KING</NAME>
      <JOB>PRESIDENT</JOB>
      <MANAGER/>
      <HIREDATE>1981-11-17T00:00:00.000-08:00</HIREDATE>
      <SALARY>5000</SALARY>
    </G_EMP>
    ...
  </LIST_G_EMP>
  <DEPTSAL>12750</DEPTSAL>
 </G_DEPT>
  <G_DEPT>
    <DEPT_NUMBER>20</DEPT_NUMBER>
    <DEPT_NAME>RESEARCH</DEPT_NAME>
    <LOCATION>DALLAS</LOCATION>
  <LIST_G_EMP>
    <G_EMP>
      <EMPLOYEE_NUMBER>7369</EMPLOYEE_NUMBER>
      <NAME>SMITH</NAME>
      <JOB>CLERK</JOB>
    ...
  </G_EMP>
  </LIST_G_EMP>
  <DEPTSAL>10875</DEPTSAL>
 </G_DEPT>
 </LIST_G_DEPT>
</dataTemplateName>

Following is the data template used to extract this data.
<?xml version="1.0" encoding="WINDOWS-1252" ?>

The template is named, an optional description
- can be provided and the default package, if any, is identified:
<dataTemplate name="Employee Listing" description="List of Employees" dataSourceRef="ORCL_DB1" defaultPackage="employee"
version="1.0">
  <parameters>- Defines a single parameter for the Department Number
  - with default of 20:
      <parameter name="p_DEPTNO" dataType="character"
defaultValue="20"/>
  </parameters>
  <dataQuery>
    <sqlStatement name="Q1">- This extracts the department information based on a
    - where clause from a pl/sql package:
        <![CDATA[SELECT DEPTNO,DNAME,LOC from dept
    where &pwhereclause
    order by deptno]]>
    </sqlStatement>
    <sqlStatement name="Q2">- This second query extracts the employee data and joins to
    - the parent query using a bind variable, :DEPTNO
        <![CDATA[SELECT EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,nvl
    (COMM,0) COMM
from EMP
    WHERE DEPTNO = :DEPTNO ]]>}
    </sqlStatement>
  </dataQuery>- A call is made to a before fetch trigger to set the
  - where clause variable in the department query, &pwhereclause:
    <dataTrigger name="beforeReport" source="employee.beforeReportTrigger"/>
  <dataStructure>- The following section specifies the XML hierarchy
  - for the returning data:
      <group name="G_DEPT" source="Q1"
      groupFilter="employee.G_DEPTFilter(:DEPT_NUMBER)">- There is a
group filter placed on the DEPT group.
- This is returned from the employee.G_DEPTFilter plsql package.
- It passes the DEPT_NUMBER value ("name" attribute) rather
  than the DEPTNO value ("value" attribute)
      <element name="DEPT_NUMBER" value="DEPTNO" />
      <element name="DEPT_NAME" value="DNAME"/>- This creates a
summary total at the department level based
- on the salaries at the employee level for each department:
      <element name="DEPTSAL" value="G_EMP.SALARY"
      function="SUM()"/>
      <element name="LOCATION" value="LOC" />
    </group>
  </group>
</dataStructure>
The PL/SQL Package:

- This is the package specification, it declares the global variables and functions contained therein

```sql
function BeforeReportTrigger return boolean;
p_DEPTNO NUMBER;
pwhereclause varchar2(3200);
function G_DEPTFilter(deptno number) return boolean;
END;
/
```

- This is the package body, it contains the code for the functions/procedures

```sql
create or replace package body employee as

- this is the event trigger called from the data template prior to the data fetch. It sets the where clause for the department query (Q1) based on the incoming data template parameter

```sql
FUNCTION BeforeReportTrigger return boolean is
begin
  IF (p_DEPTNO=10) THEN
    pwhereclause :='DEPTNO =10';
  elsif (p_DEPTNO=20) THEN
    pwhereclause:='DEPTNO =20';
  elsif (p_DEPTNO=30) THEN
    pwhereclause:='DEPTNO =30';
  elsif (p_DEPTNO=40) THEN
    pwhereclause:='DEPTNO =20';
  else
    pwhereclause:='1=1';
  end if;
end;
RETURN(TRUE);
```
- This function specifies a group filter on the Q1 group. If the department number is 30 then the data is not returned.

```sql
FUNCTION G_DEPTFilter(deptno number) return boolean is
begin
  if deptno = 30 then
    return (FALSE);
  end if;

  RETURN (TRUE);
end;
END;
/
```

General Ledger Journals Data Template Example

This data template extracts GL journals data from the E-Business Suite General Ledger schema. It is based on an existing Oracle Report that has been converted to a data template format. It follows the same format as the Employee data template but has some added functionality.
<?xml version="1.0" encoding="UTF-8" ?>
<dataTemplate name="GLRGNJ" dataSourceRef="ORA_EBS"
defaultPackage="GLRGNJ" version="1.0">
<parameters>- Parameter declaration, these will be populated at runtime.
  <parameter name="P_CONC_REQUEST_ID" dataType = "number"
    defaultValue="0"></parameter>
  <parameter name="P_JE_SOURCE_NAME" dataType="character"></parameter>
  <parameter name="P_SET_OF_BOOKS_ID" dataType="character"
    defaultValue="1"></parameter>
  <parameter name="P_PERIOD_NAME" dataType="character" Dec-97
    defaultValue=""></parameter>
  <parameter name="P_BATCH_NAME" dataType="character"></parameter>
  <parameter name="P_POSTING_STATUS" dataType="character"
    defaultValue=""></parameter>
  <parameter name="P_CURRENCY_CODE" dataType="character"
    defaultValue="USD"></parameter>
  <parameter name="P_START_DATE" dataType = "date"></parameter>
  <parameter name="P_END_DATE" dataType = "date"></parameter>
  <parameter name="P_PAGESIZE" dataType = "number"
    defaultValue="180"></parameter>
  <parameter name="P_KIND" dataType = "character"
    defaultValue="L"></parameter>
</parameters>
<lexicals>- Flexfield lexical declaration, this specifies the setup required
 - for these flexfield functions.
 - The first will return the full accounting flexfield with
   - the appropriate delimiter e.g. 01-110-6140-0000-000
<lexical type ="oracle.apps.fnd.flex.kff.select"
  name="FLEXDATA_DSP"
  application_short_name="SQLGL"
  id_flex_code="GL#"
  id_flex_num=":STRUCT_NUM"
  multiple_id_flex_num="N"
  code_combination_table_alias="CC"
  segments="ALL"
  show_parent_segments="Y"
  output_type="VALUE" />
- The second will return 'Y' if the current combination is
  - secured against the current user, 'N' otherwise
<lexical type ="oracle.apps.fnd.flex.kff.select"
  name="FLEXDATA_SECURE"
  application_short_name="SQLGL"
  id_flex_code="GL#"
  id_flex_num=":STRUCT_NUM"
  multiple_id_flex_num="N"
  code_combination_table_alias="CC"
  segments="ALL"
  show_parent_segments="Y"
  output_type="SECURITY" />
</lexicals>
<sqlStatement name="Q_MAIN">
<![CDATA[
SELECT
  S.user_je_source_name Source,
  B.name Batch_Name,
  B.default_effective_date Batch_Eff_date,
  B.posted_date Batch_Posted_Date,
]]>
</sqlStatement>
</dataQuery>
</dataTemplate>
B.je_batch_id                       Batch_Id,
B.running_total_accounted_dr       B_TOT_DR,
B.running_total_accounted_cr       B_TOT_CR,
D.je_header_id                     Header_id,
D.name                             Header_Name,
C.user_je_category_name            Category,
D.running_total_accounted_dr       H_TOT_DR,
D.running_total_accounted_cr       H_TOT_CR,
J.je_line_num                      Je_Line_Num,
decode(nvl(CC.code_combination_id, -1), -1, 'A',null)
FLEXDATA_H,
J.effective_date                   Line_Effective_Date,
J.description                      Line_Description,
J.accounted_dr                     Line_Acc_Dr,
J.accounted_cr                     Line_Acc_Cr,
D.currency_code                    Currency_Code,
D.external_reference               Header_Reference,
&POSTING_STATUS_SELECT             Recerence1_4,
nvl(J.stat_amount,0)               Line_Stat_Amount,
GLL.description                   Batch_Type,
B.actual_flag                      Actual_Flag,
GLL2.meaning                       Journal_Type,
SOB.consolidation_sob_flag        Cons_Sob_Flag,
&FLEXDATA_DSP FLEXDATA_DSP,
&FLEXDATA_SECURE FLEXDATA_SECURE
FROM gl_lookups GLL, gl_je_sources S, gl_je_categories C,
gl_je_lines J, gl_code_combinations CC, gl_je_headers D,
gl_je_batches B, gl_lookups GLL2, gl_sets_of_books SOB
WHERE GLL.lookup_code = B.actual_flag
AND GLL.lookup_type = 'BATCH_TYPE'
AND GLL2.lookup_type = 'AB_JOURNAL_TYPE'
AND GLL2.lookup_code = B.average_journal_flag
AND SOB.set_of_books_id = :P_SET_OF_BOOKS_ID
AND S.je_source_name = D.je_source
AND C.je_category_name = D.je_category
AND J.code_combination_id = CC.code_combination_id(+)
AND J.je_batch_id = D.je_batch_id
AND &CURRENCY_WHERE
AND D.je_source = NVL(:P_JE_SOURCE_NAME, D.je_source)
AND D.je_batch_id = B.je_batch_id
AND &POSTING_STATUS_WHERE
AND B.name = NVL(:P_BATCH_NAME, B.name)
AND &PERIOD_WHERE
AND B.set_of_books_id = :P_SET_OF_BOOKS_ID
ORDER BY S.user_je_source_name,
B.actual_flag,
B.name,
B.default_effective_date,
D.name,
J.je_line_num
]]>
</sqlStatement>
</dataQuery>- The original report had an AfterParameter
- and Before report triggers
<dataTrigger name="afterParameterFormTrigger" source="GLRGNJ.afterpform"/>
<dataTrigger name="beforeReportTrigger" source="GLRGNJ.beforereport"/>
<dataStructure>- A very complex XML hierarchy can be built with summary
- columns referring to lower level elements
<group name="G_SOURCE" dataType="varchar2" source="Q_MAIN">
<element name="FLEXDATA_DSP" dataType="varchar2" value="FLEXDATA_DSP" />
<element name="Line_Description" dataType="varchar2" value="Line_Description" />
<element name="Reference1_4" dataType="varchar2" value="Reference1_4" />
<element name="Line_Acc_Dr" dataType="number" value="Line_Acc_Dr" />
<element name="Line_Acc_Cr" dataType="number" value="Line_Acc_Cr" />
<element name="Line_Stat_Amount" dataType="number" value="Line_Stat_Amount" />
<element name="Line_Effective_Date" dataType="date" value="Line_Effective_Date" />
<element name="FLEXDATA_SECURE" dataType="varchar2" value="FLEXDATA_SECURE" />
</group>
</group>
</group>
</group>

<element name="R_TOT_DR" function="sum" dataType="number" value="G_SOURCE.SOU_SUM_ACC_DR" />
<element name="R_TOT_CR" function="sum" dataType="number" value="G_SOURCE.SOU_SUM_ACC_CR" />
<element name="R_TOT_STAT_AMT" function="sum" dataType="number" value="G_SOURCE.SOU_SUM_STAT_AMT" />
<element name="JE_SOURCE_DSP" function="first" dataType="number" value="G_SOURCE.Source" />
<element name="REP_BATCH_ID" function="first" dataType="number" value="G_BATCHES.Batch_Id" />
<element name="C_DATEFORMAT" dataType="varchar2" value="C_DATEFORMAT" />
</dataStructure>
- There is an after fetch trigger, this can be used to clean up
- data or update records to report that they have been reported
<dataTrigger name="afterReportTrigger" source="GLRGNJ.afterreport"/>
</dataTemplate>

**Employee XML Datasource Data Template**

This data template combines data that exists in a table called "dept" with data from an xml file called "employee.xml". It follows the same format as the Employee data template but the employee data comes from an xml file instead of from the emp table.
<?xml version="1.0" encoding="WINDOWS-1252" ?>
<dataTemplate name="Employee Listing" description="List of Employees" version="1.0">
  <parameters>
    <parameter name="p_DEPTNO" dataType="character" defaultValue="20"/>
  </parameters>
  <dataQuery>
    <sqlStatement name="Q1">
      <![CDATA[SELECT DEPTNO,DNAME,LOC from dept
                   order by deptno]]>
    </sqlStatement>
    <xml name="empxml" expressionPath=".//ROW[DEPTNO=$DEPTNO]">
      - Defines name and link to DEPTNO in Q1
      - Defines url for xml data
      - Defines name for xml hierarchy for the returning data:
        - This creates a summary total at the department level based on the salaries at the employee level for each department:
          <element name="DEPTSAL" value="G_EMP.SALARY" function="SUM()"/>
          <element name="LOCATION" value="LOC"/>
          <group name="G_EMP" source="empxml">
            <element name="EMPLOYEE_NUMBER" value="EMPNO"/>
            <element name="NAME" value="ENAME"/>
            <element name="JOB" value="JOB"/>
            <element name="MANAGER" value="MGR"/>
            <element name="HIREDATE" value="HIREDATE"/>
            <element name="SALARY" value="SAL"/>
          </group>
    </xml>
  </dataQuery>
  <dataStructure>
    - The following section specifies the XML hierarchy for the returning data:
      <group name="G_DEPT" source="Q1">
        <element name="DEPT_NUMBER" value="DEPTNO" />  
        <element name="DEPT_NAME" value="DNAME"/>
      </group>
  </dataStructure>
</dataTemplate>
Creating an RTF Template

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.

BI 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 BI Publisher will recognize and maintain the design.

During design time, you add data fields and other markup to your template using BI 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, BI Publisher also supports the use of pure XSL elements in the template.

In addition to your word processing application's formatting features, BI 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 BI Publisher.

Supported Modes

BI 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 BI 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. BI Publisher supports Microsoft Word 2000 (or later) with Microsoft Windows version 2000 (or later).

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

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

**Prerequisites**

Before you design your template, you must:

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

**Overview**

Creating an RTF template file consists of two basic steps:

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

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

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

**Using the BI Publisher Template Builder**

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

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

- Insert data fields
Creating an RTF Template

- 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

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

Note the following:
- The data fields that are defined on the template
  For example: Supplier, Invoice Number, and Invoice Date
- The elements of the template that will repeat when the report is run.
  For example, all the fields on the template will repeat for each Supplier that is
reported. Each row of the invoice table will repeat for each invoice that is reported.

XML Input File

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

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

```xml
<?xml version="1.0" encoding="WINDOWS-1252" ?>
- <VENDOR_REPORT>
  - <LIST_G_VENDOR_NAME>
    - <G_VENDOR_NAME><VENDOR_NAME>COMPANY A</VENDOR_NAME>
    - <LIST_G_INVOICE_NUM>
      - <G_INVOICE_NUM>
        <SET_OF_BOOKS_ID>124</SET_OF_BOOKS_ID>
        <GL_DATE>10-NOV-03</GL_DATE>
        <INV_TYPE>Standard</INV_TYPE>
        <INVOICE_NUM>031110</INVOICE_NUM>
        <INVOICE_DATE>10-NOV-03</INVOICE_DATE>
        <INVOICE_CURRENCY_CODE>EUR</INVOICE_CURRENCY_CODE>
        <ENT_AMT>122</ENT_AMT>
        <ACCTD_AMT>122</ACCTD_AMT>
        <VAT_CODE>VAT22%</VAT_CODE>
      </G_INVOICE_NUM>
    </LIST_G_INVOICE_NUM>
  </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:** BI 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 7-81.

placeholders

to designate the repeating elements, you define groups.

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 BI Publisher that for each occurrence of an element (parent), you want the included fields (children) displayed. At runtime, BI 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 7-15.

For a detailed list of supported formatting features in Microsoft Word, see Supported
Adding Markup to the Template Layout

BI 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.

BI Publisher provides tags to add markup to your template.

Note: For the XSL equivalents of the BI Publisher tags, see XSL Equivalent Syntax, page 8-15.

Creating Placeholders

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

Enter placeholders in your document using the following syntax:

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

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

There are two ways to insert placeholders in your document:

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

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

Basic RTF Method

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

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

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

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

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

```
<table>
<thead>
<tr>
<th>Invoice Num</th>
<th>Invoice</th>
</tr>
</thead>
<tbody>
<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 BI 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 7-12 for the completed basic RTF markup.

**Defining Groups**

By defining a group, you are notifying BI Publisher that for each occurrence of an element, you want the included fields displayed. At runtime, BI 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.
If you insert the grouping tags around text or formatting elements, the text and formatting elements between the group tags will be repeated.

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

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

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

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

**Basic RTF Method**

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

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

<?for-each:G_VENDOR_NAME?>

before the Supplier field that you previously created.

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

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

Form Field Method

1. Insert a form field to designate the beginning of the group.
   In the help text field enter:
   `<?for-each:group element tag name?>`
   To create the Suppliers group in the example, insert a form field before the Suppliers field that you previously created. In the help text field enter:
   `<?for-each:G_VENDOR_NAME?>`
   For the example, enter the Default text "Group: Suppliers" to designate the beginning of the group on the template. The Default text is not required, but can make the template easier to read.

2. Insert a form field after the final placeholder element in the group. In the help text field enter `<?end for-each?>`.
   For the example, enter the Default text "End: Suppliers" after the summary row to designate the end of the group on the template.
   The following figure shows the template after the markup to designate the Suppliers group was added.
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

BI Publisher supports the use of the native RTF header and footer feature. To create a header or footer, use your word processing application's header and footer insertion tools. As an alternative, or if you have multiple headers and footers, you can use \texttt{\textbf{start:body}} and \texttt{\textbf{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 \texttt{\textbf{start body}}/\texttt{\textbf{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 BI 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:

\texttt{\textbf{<\texttt{\textbf{start:body}}>}}

\texttt{\textbf{<\texttt{\textbf{end body}>}}}

---

Company Confidential
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.
Inserting Images and Charts

Images

BI 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'}

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, page 6-1) 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:

```xml
<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`)
- `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. BI 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:

```xml
<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:

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

or in centimeters:

```xml
<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:

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

**Chart Support**

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

**Adding a Sample Chart**

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

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

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

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

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

**Inserting the Dummy Image**

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

The following figure shows an example of a dummy image:

![Graph](image.png)

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 BI 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:

```xml
<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>
```

Web browsers display alternative text while pictures are loading or if they are missing. Web search engines use alternative text to help find Web pages.
The first element of your chart text must be the `chart:` element to inform the RTF parser that the following code describes a chart object.

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

The following code section defines the chart type and attributes:

```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>
```

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:

```xml
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:

```xml
<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>
  </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](chart.png)

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:

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

The following code hides the chart grid:

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

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

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

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

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

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

```xml
<Title text="Company Sales 2004" visible="true">
  <GraphFont name="Tahoma" bold="false"/>
</Title>
```
**Drawing, Shape, and Clip Art Support**

BI 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 7-53.

**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

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

Microsoft Equation

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

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

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, BI 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 7-33. 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:

```xml
<?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:

```xml
<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 Diagram](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 Diagram]

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, BI 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 BI Publisher’s page break alias. See Special Features: Page Breaks, page 7-48.
• 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

BI 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
  
  Note: This feature is not supported for RTF output.

  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

BI 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:
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 7-42.

This template will produce the following multicolumn output:

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

Background and Watermark Support

BI 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:

![Fill Effects dialog](image)

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:

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

Example

For the following XML, assume you want to create a page break for each new supplier:
In the template sample shown in the following figure, the field called PageBreak contains the split-by-page-break syntax:

<table>
<thead>
<tr>
<th>Invoice Number</th>
<th>Invoice Date</th>
<th>Amount</th>
<th>Running Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>10001-1</td>
<td>1-Jan-2005</td>
<td>100.00</td>
<td>100.00</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. BI 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:pagenumber?>`

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

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

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

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

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

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

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

Enter the following in your template:

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

**Note:** You must first declare the parameter in your template. See Defining Parameters in Your Template, page 7-88.

### Last Page Only Content

BI 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. BI 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.

BI 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:

   ```
   <?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>
  <VENDOR>
    ...
  </VENDOR>
  ...
  <INVOICE>
    ...
  </INVOICE>
  <SUMMARY>
    <SUM_ENT_AMT>61435</SUM_ENT_AMT>
    <SUM_ACCTD_AMT>58264.68</SUM_ACCTD_AMT>
    <TAX_CODE>EU22%</TAX_CODE>
  </SUMMARY>
</INVOICELIST>
```

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

**Tax Summary**

<table>
<thead>
<tr>
<th>Tax Code</th>
<th>Entered Amount</th>
<th>Accounted Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAT 18.5</td>
<td>100</td>
<td>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 contain headers and footers that you want to carry over onto the last page, you must reinsert them on the last page. For more information about headers and footers see Defining Headers and Footers, page 7-15.

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 BI 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 7-16) 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

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

1. Select the text or shape.

2. Use the right-mouse menu to select **Hyperlink**; or, select **Hyperlink** from the **Insert** menu.

3. Enter the URL using any of the methods provided on the **Insert Hyperlink** dialog box.

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

![Insert Hyperlink Dialog Box](image)

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

{URL_LINK}

where URL_LINK is the incoming data element name.

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

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

where PRODUCT_NAME is the incoming data element name.

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

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

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

\[\{\text{SERVER\_URL}\}\{\text{REPORT}\}/\text{cstid=\{CUSTOMER\_ID\}}\]

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


**Inserting Internal Links**

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

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

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

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

4. Select the text or shape in your document that you want to link back to the
Bookmark target.

5. Use the right-mouse menu to select **Hyperlink**; or select **Hyperlink** from the **Insert** menu.

6. On the **Insert Hyperlink** dialog, select **Bookmark**.

7. Choose the bookmark you created from the list.

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

**Table of Contents**

BI 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.

BI 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 7-56.

- To create links for a static table of contents:

  Enter the syntax:

  `<copy-to-bookmark:>`

  directly above your table of contents and

  `<end copy-to-bookmark:>`
• 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**

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

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

To create the index for the continent example:

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

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

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

<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>
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>7,390,000</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>

Where the fields have the following values:

<table>
<thead>
<tr>
<th>Field</th>
<th>Form Field Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FE</td>
<td>&lt;?for-each:country?&gt;</td>
<td>Begins the 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;100000?</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;?continentIndex?&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. BI Publisher supports the usage of simple "if" statements, as well as more complex "choose" expressions.

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

**If Statements**

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

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

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

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

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

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

   ![If Statements](image)

**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 BI 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 7-121.

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 BI 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**

BI 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:

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

You can also nest these statements as follows:

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

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

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

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

**Choose Statements**

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

Use the following syntax for these elements:

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

**"Choose" Conditional Formatting Example**

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

In the following figure, the form field default text is displayed. The form field help text entries are shown in the table following the example.
Column 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
      VAT 15%  |  1000  |  1000  |  1000  |  1000  |  1000
      Otherwise
      VAT 15%  |  1000  |  1000  |  1000  |  1000  |  1000
    </When>
  </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>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plasma TV</td>
<td>4000</td>
</tr>
<tr>
<td>DVD Player</td>
<td>300</td>
</tr>
<tr>
<td>VCR</td>
<td>200</td>
</tr>
<tr>
<td>Receiver</td>
<td>350</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Default Text</th>
<th>Form Field Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grp:Item</td>
<td>&lt;?for-each:item?&gt;</td>
<td>Holds the opening for-each loop for the item element.</td>
</tr>
<tr>
<td>Plasma TV</td>
<td>&lt;?name?&gt;</td>
<td>The placeholder for the name element from the XML file.</td>
</tr>
</tbody>
</table>
Default Text | Form Field Entry | Description
--- | --- | ---
IF | `<?if@column:/items/@type="PRIVATE"?>` | The opening of the if statement to test for the attribute value "PRIVATE". Note that this syntax uses an XPath expression to navigate back to the "items" level of the XML to test the attribute. For more information about using XPath in your templates, see XPath Overview, page 7-118.

Quantity | N/A | Boilerplate heading
end-if | `<?end if?>` | Ends the if statement.
20 | `<?if@column:/items/@type="PRIVATE"?><?quantity?><?end if?>` | The placeholder for the quantity element surrounded by the "if" statement.
1,000.00 | `<?price?>` | The placeholder for the price element.
end grp | `<?end for-each?>` | Closing tag of the for-each loop.

The conditional column syntax is the "if" statement syntax with the addition of the @column clause. It is the @column clause that instructs BI 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 7-121.

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**

BI 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.

```
<for-each SALE>
  <if big>
    <INDUSTRY>
      <YEAR>
      <MONTH>
      <SALES>
    </SALES>
  </if>
</for-each>
```

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>
Creating an RTF Template

Conditionally Highlighting a Row

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

<table>
<thead>
<tr>
<th>Industry</th>
<th>Year</th>
<th>Month</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>for-each SALE</td>
<td>Format;</td>
<td>INDUSTRY</td>
<td>YEAR</td>
</tr>
</tbody>
</table>

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

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

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>DE Account1-232-4444</td>
<td>CH100.00</td>
<td>CH2100.00</td>
</tr>
</tbody>
</table>

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

Note: For more information about context commands, see Using Context Commands, page 7-121.
The code to highlight the debit column as shown in the table is:

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

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

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

The xdofo:ctx component is an BI 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

BI 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 output 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.

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

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

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

where

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

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

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

```xml
<?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...
symbols, see Using the Oracle Format Mask, page 7-110.

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

This example uses the following XML:

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

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

<table>
<thead>
<tr>
<th>Debit</th>
<th>Credit</th>
</tr>
</thead>
<tbody>
<tr>
<td>FE 100.00</td>
<td>90.00 Net</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. 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. 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>
</tbody>
</table>
Default Text Entry | Form Field Help Text Entry | Description
---|---|---
Net | `<add-page-total:net;'debit - credit'?>` | Creates a net page total by subtracting the credit values from the debit values.
EFE | `<end for-each?>` | Closes the for-each loop.

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

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

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

**Page Total Debit:** 750.00

**Page Total Credit:** 390.00

**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>Inv</th>
<th>Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>1-Jan-05</td>
<td>100</td>
<td>1004</td>
<td>1-Jan-05</td>
<td>100</td>
</tr>
<tr>
<td>1002</td>
<td>1-Jan-05</td>
<td>100</td>
<td>1005</td>
<td>1-Jan-05</td>
<td>100</td>
</tr>
<tr>
<td>1003</td>
<td>1-Jan-05</td>
<td>100</td>
<td>1006</td>
<td>1-Jan-05</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>300</td>
<td></td>
<td></td>
<td>600</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 7-72 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>&lt;?init-page-total: InvAmt?&gt;</td>
<td>Declares &quot;InvAmt&quot; as the placeholder that will hold the page total.</td>
</tr>
<tr>
<td>FE</td>
<td>&lt;?for-each:INVOICE?&gt;</td>
<td>Begins the INVOICE group.</td>
</tr>
<tr>
<td>100.00</td>
<td>&lt;?INVAMT?&gt;</td>
<td>Placeholder for the Invoice Amount tag.</td>
</tr>
<tr>
<td>InvAmt</td>
<td>&lt;?add-page-total:InvAmt;INVAMT?&gt;</td>
<td>Assigns the &quot;InvAmt&quot; page total object to the INVAMT element in the data.</td>
</tr>
<tr>
<td>EFE</td>
<td>&lt;?end for-each?&gt;</td>
<td>Closes the INVOICE group.</td>
</tr>
<tr>
<td>End PTs</td>
<td>&lt;?end-page-total:InvAmt?&gt;</td>
<td>Closes the &quot;InvAmt&quot; page total.</td>
</tr>
</tbody>
</table>

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

```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 7-110.

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. BI 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 7-15 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 7-87) 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>
    <INVNUNM>10001-1</INVNUNM>
    <INVDATDE>1-Jan-2005</INVDATDE>
    <INVAMT>100</INVAMT>
  </INVOICE>
  <INVOICE>
    <INVNUNM>10001-2</INVNUNM>
    <INVDATDE>10-Jan-2005</INVDATDE>
    <INVAMT>200</INVAMT>
  </INVOICE>
  <INVOICE>
    <INVNUNM>10001-1</INVNUNM>
    <INVDATDE>11-Jan-2005</INVDATDE>
    <INVAMT>150</INVAMT>
  </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>

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

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

Sorting

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

`<?sort:element name?>`

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

`<?sort:VENDOR_NAME?>`

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


Checking for Nulls

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

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

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

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

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

- To define behavior when the element is not present, use the following:
  
  `<?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:
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, BI Publisher will loop through the occurrences of the new groupings, displaying the fields that you defined in your template.

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

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

**Template Example**

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

<table>
<thead>
<tr>
<th>Group by Country</th>
<th>Country USA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Group by Year</td>
</tr>
<tr>
<td></td>
<td>Year 2000</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Title</td>
</tr>
<tr>
<td>Group Details</td>
<td>My CD</td>
</tr>
<tr>
<td>Group Details</td>
<td>My CD</td>
</tr>
<tr>
<td>End Group by Year</td>
<td></td>
</tr>
<tr>
<td>End Group by Country</td>
<td></td>
</tr>
</tbody>
</table>

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

```
Country: USA

Year: 1985

<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>

Country: UK

Year: 1988

<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>

Year: 1990

<table>
<thead>
<tr>
<th>Title</th>
<th>Artist</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Still got the blues</td>
<td>Gary Moore</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:

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

**Example**

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

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

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

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

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

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

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

Annual Temperature Summary

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

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

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

Note the following about the form field tags:

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

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

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

- The <?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.

### Using Variables

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

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

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

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

Use the following syntax to retrieve a variable value:

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

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

```xml
<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.

See the section on Running Totals, page 7-78 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: For BI Publisher Enterprise users, all name-value parameter pairs are passed to the template. You must register the parameters that you wish to utilize in your template using the syntax described below.

Using a parameter in a template

1. Declare the parameter in the template.

   Use the following syntax to declare the parameter:

   ```xml
   <?param@begin:parameter_name;parameter_value?>
   ```

   where

   - `parameter_name` is the name of the parameter
   - `parameter_value` is the default value for the parameter (the parameter_value is optional)
   - `param@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. If you are not using BI Publisher Enterprise, but only the core libraries:

   At runtime, pass the parameter to the BI Publisher engine programmatically.

   Prior to calling the FOPProcessor API 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");
   ```

   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:
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](image)

<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;?param@begin:InvThresh?&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>
<tr>
<td>EFE</td>
<td>&lt;?end for-each?&gt;</td>
<td>Closing tag for the for-each loop.</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**

BI Publisher properties that are available in the BI Publisher Configuration file can alternatively be embedded into the RTF template. The properties set in the template are resolved at runtime by the BI 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 BI Publisher Configuration File, *Oracle Business Intelligence Publisher Administrator’s and Developer’s Guide* for more information about the BI 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 BI Publisher PDF security properties and obtain the password value from the incoming XML data.

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

- **Name** - enter the BI Publisher property name prefixed with "xdo-

- **Type** - select "Text"

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

The following figure shows the Properties dialog:
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 BI Publisher where to find the font at runtime. You tell BI Publisher where to find the font by setting the "font" property. Assume the font is located in "/tmp/fonts", then you would enter the following in the Properties dialog:

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

**Type:** Text

**Value:** truetype./tmp/fonts/XMLPScript.ttf

When the template is applied to the XML data on the server, BI 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 Business Intelligence Publisher Administrator’s and Developer’s Guide*.

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:
<PO>
  <security>true</security>
  <password>welcome</password>
  <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 7-88.

**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
Creating an RTF Template

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.

BI 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 BI 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 7-98.</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 7-81) 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 7-121.</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>

<table>
<thead>
<tr>
<th>Industry</th>
<th>2005</th>
<th>2004</th>
<th>2003</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motor Vehicle Dealers</td>
<td>3000</td>
<td>6000</td>
<td>1200</td>
</tr>
<tr>
<td>Home Furnishings</td>
<td>3200</td>
<td>7770</td>
<td>3300</td>
</tr>
<tr>
<td>Electronics</td>
<td>9000</td>
<td>9000</td>
<td>4300</td>
</tr>
<tr>
<td>Food and Beverage</td>
<td>1200</td>
<td>900</td>
<td>5600</td>
</tr>
</tbody>
</table>
Creating an RTF Template

Default Text Entry | Form Field Help Text | Description
--- | --- | ---

for: | `<?for-each-group@cell:current-group();YEAR?>` | Uses the regrouping syntax (see Regrouping the XML Data, page 7-81) to group the data by YEAR; and the `<?cell>` context command to create a table cell for each group (YEAR).

sum(Sales) | `<?sum(current-group()//SALES)?>` | Sums the sales for the current group (YEAR).

end | `<?end for-each-group?>` | Closes the for-each-group statement.

end | `<?end for-each-group?>` | Closes the for-each-group statement.

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 7-121.)

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 know how many ranges will have data to report. You can define a dynamic data column to split into the correct number of columns at runtime.

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

- Dynamic Column Header
  `<?split-column-header:group element name?>`
  Use this tag to define which group to split for the column headers of a table.

- Dynamic Column `<?split-column-data:group element name?>`
  Use this tag to define which group to split for the column data of a table.

- Dynamic Column Width
  `<?split-column-width:name?>` or `<?split-column-width:@width?>`
Use one of these tags to define the width of the column when the width is described in the XML data. The width can be described in two ways:

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

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

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

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

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

Using the dynamic column tags in form fields, set up the table in two columns as shown
in the following figure. The first column, "Test Score" is static. The second column,
"Column Header and Splitting" is the dynamic column. At runtime this column will
split according to the data, and the header for each column will be appropriately
populated. The Default Text entry and Form Field Help entry for each field are listed in
the table following the figure. (See Form Field Method, page 7-8 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.
Creating an RTF Template

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

BI 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 7-126.

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

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

**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":

---

7-104  Oracle Business Intelligence 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 |
| C      | Returns the ISO currency symbol in the specified position. |
Symbol | Meaning
---|---
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**

BI 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
date_format

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 7-113 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>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>Symbol</td>
<td>Meaning</td>
</tr>
<tr>
<td>--------</td>
<td>---------</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>ffff</td>
<td>Displays seconds fractions represented in four digits.</td>
</tr>
<tr>
<td>fffff</td>
<td>Displays seconds fractions represented in five digits.</td>
</tr>
<tr>
<td>ffffff</td>
<td>Displays seconds fractions represented in six digits.</td>
</tr>
<tr>
<td>ffffffff</td>
<td>Displays seconds fractions represented in seven digits.</td>
</tr>
</tbody>
</table>
### Symbol Meaning

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>tt</td>
<td>The AM/PM designator defined in AMDesignator or PMDesignator, if any.</td>
</tr>
<tr>
<td>z</td>
<td>Displays the time zone offset for the system’s current time zone in whole hours only. (This element can be used for formatting only)</td>
</tr>
<tr>
<td>zz</td>
<td>Displays the time zone offset for the system’s current time zone in whole hours only. (This element can be used for formatting only)</td>
</tr>
<tr>
<td>zzz</td>
<td>Displays the time zone offset for the system’s current time zone in hours and minutes.</td>
</tr>
<tr>
<td>:</td>
<td>The default time separator defined in TimeSeparator.</td>
</tr>
<tr>
<td>/</td>
<td>The default date separator defined in DateSeparator.</td>
</tr>
<tr>
<td>'</td>
<td>Quoted string. Displays the literal value of any string between two ` ' characters.</td>
</tr>
<tr>
<td>&quot;</td>
<td>Quoted string. Displays the literal value of any string between two ` &quot; characters.</td>
</tr>
</tbody>
</table>

### Using the Oracle Format Mask

To apply the Oracle format mask to a date field:

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

   ```xml
   <?format-date:date_string;
   'ABSTRACT_FORMAT_MASK';'TIMEZONE'?>
   or
   <?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:
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></td>
</tr>
<tr>
<td>,</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
</tr>
<tr>
<td>:</td>
<td></td>
</tr>
<tr>
<td>&quot;text&quot;</td>
<td></td>
</tr>
<tr>
<td>AD</td>
<td>AD indicator with or without periods.</td>
</tr>
<tr>
<td>A.D.</td>
<td></td>
</tr>
<tr>
<td>AM</td>
<td>Meridian indicator with or without periods.</td>
</tr>
<tr>
<td>A.M.</td>
<td></td>
</tr>
<tr>
<td>BC</td>
<td>BC indicator with or without periods.</td>
</tr>
<tr>
<td>B.C.</td>
<td></td>
</tr>
<tr>
<td>CC</td>
<td>Century. For example, 2002 returns 21; 2000 returns 20.</td>
</tr>
<tr>
<td>DAY</td>
<td>Name of day, padded with blanks to length of 9 characters.</td>
</tr>
<tr>
<td>D</td>
<td>Day of week (1-7).</td>
</tr>
<tr>
<td>DD</td>
<td>Day of month (1-31).</td>
</tr>
<tr>
<td>DDD</td>
<td>Day of year (1-366).</td>
</tr>
<tr>
<td>DL</td>
<td>Returns a value in the long date format.</td>
</tr>
<tr>
<td>DS</td>
<td>Returns a value in the short date format.</td>
</tr>
<tr>
<td>DY</td>
<td>Abbreviated name of day.</td>
</tr>
<tr>
<td>Symbol</td>
<td>Meaning</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>E</td>
<td>Abbreviated era name.</td>
</tr>
<tr>
<td>EE</td>
<td>Full era name.</td>
</tr>
</tbody>
</table>
| FF[1..9]| 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.  
Example:  
'HH:MI:SS,FF3' |
| HH     | Hour of day (1-12). |
| HH12   | Hour of day (1-12). |
| HH24   | Hour of day (0-23). |
| MI     | Minute (0-59). |
| MM     | Month (01-12; JAN = 01). |
| MON    | Abbreviated name of month. |
| MONTH  | Name of month, padded with blanks to length of 9 characters. |
| PM     | Meridian indicator with or without periods. |
| P.M.   | |
| RR     | Lets you store 20th century dates in the 21st century using only two digits. |
| RRRR   | 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. |
| SS     | Seconds (0-59). |
| 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) |
### Symbol Meaning

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TZH</td>
<td>Time zone hour. (See TZM format element.)</td>
</tr>
<tr>
<td>TZM</td>
<td>Time zone minute. (See TZH format element.)</td>
</tr>
<tr>
<td>Example:</td>
<td></td>
</tr>
<tr>
<td>‘HH:MI:SS.FFTZH:TZM’</td>
<td></td>
</tr>
<tr>
<td>TZR</td>
<td>Time zone region information. The value must be one of the time zone regions supported in the database. Example: PST (Pacific Standard Time)</td>
</tr>
<tr>
<td>WW</td>
<td>Week of year (1-53) where week 1 starts on the first day of the year and continues to the seventh day of the year.</td>
</tr>
<tr>
<td>W</td>
<td>Week of month (1-5) where week 1 starts on the first day of the month and ends on the seventh.</td>
</tr>
<tr>
<td>X</td>
<td>Local radix character.</td>
</tr>
<tr>
<td>YYYY</td>
<td>4-digit year.</td>
</tr>
<tr>
<td>YY</td>
<td>Last 2, or 1 digit(s) of year.</td>
</tr>
<tr>
<td>Y</td>
<td></td>
</tr>
</tbody>
</table>

### Default Format Mask

If you do not want to specify a format mask with either the MS method or the Oracle method, you can omit the mask definition and use the default format mask. The default format mask is the MEDIUM abstract format mask from Oracle. (See Oracle Abstract Format Masks, page 7-113 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:

`<?format-date:hiredate?>`

### 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>

Calendar and Timezone 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 (XDO_CALENDAR_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**

BI 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 BI 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:

<table>
<thead>
<tr>
<th>Normal</th>
<th>GnuMICR</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>·</td>
<td>·</td>
</tr>
<tr>
<td></td>
<td>·</td>
<td>·</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

   formfield method

   6 5 4 5 5 4 7 4 7 4 7 4 4 text method

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

3. Set the BI Publisher "font" property.
   You can set the font property for the report in the BI Publisher Font Mappings page, or in the configuration file.

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

   ```xml
   <font family="MyFontName" style="normal" weight="normal">
     <truetype path="\user\fonts\MyFontName.ttf"/>
   </font>
   
   See BI Publisher Configuration File, *Oracle Business Intelligence Publisher Administrator’s and Developer’s Guide* for more information.

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

   Now you can run your report and BI Publisher will use the font in the output as designed. For PDF output, the advanced font handling features of BI 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.

Advanced Barcode Formatting

BI 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 BI 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 Business Intelligence Publisher Administrator’s 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 BI 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:

```xml
<?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

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](http://www.w3.org/TR/xpath)

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

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

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

```xml
//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:

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

This example will return the following XML:

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

Further limit your search by using square brackets. The brackets locate elements with certain child nodes or specified values. For example, the following expression locates all CDs recorded by Bob Dylan:
Or, if each CD element did not have an PRICE element, you could use the following expression to return only those CD elements that include a PRICE element:

/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 catttype="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:

/CD[1]

The sample returns the first CD element:

<CD catttype="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:

/CD[

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:

/CD[YEAR >=1985]

**Starting Reference**

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

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

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

Context and Parent

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

../*

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

/CATALOG/CD/..

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

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

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

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

Namespace Support

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

<?namespace:namespace name= namespace url?>

For example:


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

<?fsg:ReportName?>

Using the Context Commands

The BI 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.
BI 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, BI 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 BI Publisher syntax and those using native XSL.

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

- To specify a context for an XSL command, add the xdofo:ctx="context" attribute to your tags to specify the context for the insertion of the instructions. The value of the context determines where your code is placed.
  
  For example:
  ```xml
  <xsl:for-each xdofo:ctx="section" select ="INVOICE">
    <xsl:attribute xdofo:ctx="inblock"
      name="background-color">red</xsl:attribute>
  </xsl:for-each>
  ```

BI Publisher supports the following context types:

<table>
<thead>
<tr>
<th>Context</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>section</td>
<td>The statement affects the whole section including the header and footer. For example, a <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 7-92 for an example of this usage.</td>
</tr>
</tbody>
</table>
Context Description

**column**  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 7-65 for an example.

**cell**  The statement will affect the cell of a table. This is often used together with @column in cross-tab tables to create a dynamic number of columns.

See Cross-Tab Support, page 7-94 for an example.

**block**  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 7-70 for an example.

**inline**  The context will become the single statement inside an fo:inline block. This context is used for variables.

**incontext**  The statement is inserted immediately after the surrounding statement. This is the default for <?sort?> statements that need to follow the surrounding for-each as the first element.

**inblock**  The statement becomes a single statement inside an fo:block (RTF paragraph). This is typically not useful for control statements (such as if and for-each) but is useful for statements that generate text, such as call-template.

**inlines**  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 7-62.

**begin**  The statement will be placed at the beginning of the XSL stylesheet. This is required for global variables. See Defining Parameters, page 7-88.

**end**  The statement will be placed at the end of the XSL stylesheet.

The following table shows the default context for the BI Publisher commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply-template</td>
<td>inline</td>
</tr>
</tbody>
</table>

Creating an RTF Template  7-123
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. BI 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 BI 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">`

**BI 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"/>
BI 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"/>
BI 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"/>
BI Publisher Tag: `<?template:name?>`

Variable Declaration
Use this element to declare a local or global variable.

XSL Syntax: `<xsl:variable name="name"/>
BI Publisher Tag: `<?variable:name?>

Example:
```xml
<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"/>
BI 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">`

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

**Note:** The namespace must be declared in the template. See Namespace Support, page 7-121.

### 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 BI Publisher can be found in the Appendix: Supported XSL-FO Elements, page A-1.

**Guidelines for Designing RTF Templates for Microsoft PowerPoint Output**

BI Publisher can generate your RTF template as PowerPoint output enabling you to get report data into your key business presentations. Currently, the PowerPoint document generated is a simple export of the formatted data and charts to PowerPoint.

**Limitations**

Following are limitations when working with PowerPoint as an output:

- When designing tables for your PowerPoint slide, you must define the table border type as a single line (double border, dash, and other types are not supported).
- Hyperlinks are not supported.
- Charts and graphs are inserted as PNG images (SVG is not supported).
- Shapes are not supported.
- Text position may be slightly incorrect if you use right-align.
- Paper size must be the same on all pages of the RTF template. You cannot have mixed paper sizes in the same document.
- Bidirectional languages are not supported.
- Text position may be slightly incorrect for Chinese, Japanese, and Korean fonts when using bold or italic effects on characters. This is because Microsoft uses bold or italic emulation when there is no bold or italic font.
- All Unicode languages, except bidirectional languages, are supported.
- BI Publisher’s font fallback mechanism is not supported for PowerPoint templates. If you wish to use a font that is not installed, ensure that you have configured it with BI Publisher.

**Usage Guidelines**

Following are guidelines to help you when designing an RTF template intended for PowerPoint output:
• PowerPoint output will preserve the page orientation (portrait or landscape) defined in the RTF template. Most presentations are oriented in landscape so this is the recommended orientation of your RTF template.

• A page break in your RTF template will generate a new slide.

• The background color of the slides are always generated as white. If you prefer a different background color, you must change the color after the PowerPoint file is generated.

**Configuring Fonts for the BI Publisher Server**

Support for PowerPoint output does not include the font fallback mechanism that is used for other types of output in BI Publisher. If you are using a nonstandard font in your template, you must configure the BI Publisher server for each font used in the RTF template for generating PowerPoint output. You will need to copy these fonts to your BI Publisher Server and define the Font Mappings for RTF templates. This can be done for the entire system or for individual reports. See Defining Font Mappings, page 13-15 for more details.

C:\Program Files\Oracle\BI Publisher Desktop\Template Builder for Word\config

**Configuring Fonts for the BI Publisher Template Builder**

When using the BI Publisher Template Builder to design your report, to correctly preview PPT output that uses non-English or non-standard fonts, you will need to define the fonts in the BI Publisher configuration file. This configuration file is called xdo.cfg and is typically found in:

C:\Oracle\BI Publisher\BI Publisher Desktop\Template Builder for Word\config\xdo.cfg

Note that if you have not used this file yet you may find the file "xdo example.cfg" instead. This file must be saved with an encoding of UTF-8 and provide a full and absolute path for each font defined. Otherwise, you will encounter issues such as characters overlays and wrapping that does not work.

1. Navigate to C:\Oracle\BI Publisher\BI Publisher Desktop\Template Builder for Word\config\xdo.cfg

2. Open the xdo.cfg file and update the font mappings. For information on updating font mappings directly in the xdo.cfg file, see Font Definitions, *Oracle Business Intelligence Publisher Administrator’s and Developer’s Guide*.

3. Save the xdo.cfg in UTF-8 format.

The following figure shows a sample xdo.cfg file:
Creating an RTF Template

```xml
<config version="1.0.0" xmlns="http://xmlces.xeoleo.com/cep/config/m"
</config>
```
BI Publisher has extended a set of SQL and XSL functions for use in RTF templates. The syntax for these extended functions is

```
<?xdofx:expression?>
```

for extended SQL functions or

```
<?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:**

```
<?xdofx:rpad(concat(FIRST_NAME,LAST_NAME),30,'x')?>
```

because `concat` is an XSL expression. Instead, you could use the following:

**CORRECT:**

```
<?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;?xdo fx:2+3?&gt;</td>
<td>Addition</td>
</tr>
<tr>
<td>2-3</td>
<td>&lt;?xdo fx:2-3?&gt;</td>
<td>Subtraction</td>
</tr>
<tr>
<td>2*3</td>
<td>&lt;?xdo fx:2*3?&gt;</td>
<td>Multiplication</td>
</tr>
<tr>
<td>2/3</td>
<td>&lt;?xdo fx:2/3?&gt;</td>
<td>Division</td>
</tr>
<tr>
<td>2**3</td>
<td>&lt;?xdo fx: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;?xdo fx: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;?xdo fx: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>
<tr>
<td><code>trim()</code></td>
<td><code>&lt;?xdoxslt:trim(' a ')?&gt;</code></td>
<td>Removes spaces in a string. Enter the text to be trimmed, the function returns the trimmed text.</td>
</tr>
<tr>
<td><code>ltrim()</code></td>
<td><code>&lt;?xdoxslt:ltrim(' a ')?&gt;</code></td>
<td>Removes the leading white spaces in a string.</td>
</tr>
<tr>
<td><code>rtrim()</code></td>
<td><code>&lt;?xdoxslt:rtrim(' a ')?&gt;</code></td>
<td>Removes the trailing white spaces in a string.</td>
</tr>
<tr>
<td><code>truncate</code></td>
<td><code>&lt;?xdoxslt:truncate (number [, integer ] )?&gt;</code></td>
<td>The <code>truncate</code> function returns <code>number</code> truncated to <code>integer</code> places right of the decimal point. If <code>integer</code> is omitted, then <code>number</code> is truncated to the whole integer value. <code>integer</code> can be negative to truncate values left of the decimal point. <code>integer</code> must be an integer. Example: <code>&lt;?xdoxslt:truncate(-2.3333)?&gt;</code> returns -2. Example: <code>&lt;?xdoxslt:truncate(2.7777, 2)?&gt;</code> returns 2.77. Example: <code>&lt;?xdoxslt:truncate(27.7777, -1)?&gt;</code> returns 20.</td>
</tr>
<tr>
<td><code>replicate</code></td>
<td><code>&lt;?xdoxslt:replicate('string ', integer)?&gt;</code></td>
<td>The <code>replicate</code> function will replicate the specified string the specified number of times. Example: <code>&lt;?xdoxslt:replicate('oracle', 3)?&gt;</code> returns oracleoracleoracle.</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')` | `<xdofx:decode('xxx','bbb','ccc','xxx','ddd')>` | The `decode` function has the functionality of an IF-THEN-ELSE statement. The syntax for the `decode` function is:  

```sql
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)` | `<xdofx:Instr('abcabcabc','a',2)?>` | The `instr` function returns the location of a substring in a string. The syntax for the `instr` function is:  

```sql
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 from the end of `string1` and then searches towards the beginning of `string1`.
- `nth appearance` is the nth appearance of `string2`. |
| `substr('abcdefg',2,3)` | `<xdofx:substr('abcdefg',2,3)?>` | The `substr` function allows you to extract a substring from a string. The syntax for the `substr` function is:  

```sql
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. |
<table>
<thead>
<tr>
<th>SQL Statement or XSL Expression</th>
<th>Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>left</strong></td>
<td><code>&lt;?xdoxslt:left('abcdefg', 3)&gt;</code></td>
<td>Enables you to extract the specified number of characters from a string, starting from the left. The syntax is <code>left(string, Numchars)</code> For example, <code>&lt;?xdoxslt:left('abcdefg', 3)&gt;</code> returns <code>abc</code></td>
</tr>
<tr>
<td><strong>right</strong></td>
<td><code>&lt;?xdoxslt:right('abcdefg', 3)&gt;</code></td>
<td>Enables you to extract the specified number of characters from a string, starting from the right. The syntax is <code>right(string, Numchars)</code> For example, <code>&lt;?xdoxslt:right('abcdefg', 3)&gt;</code> returns <code>efg</code></td>
</tr>
<tr>
<td><strong>replace(name,'John','Jon')</strong></td>
<td><code>&lt;?xdofx:replace(name,'John','Jon')&gt;</code></td>
<td>The replace function replaces a sequence of characters in a string with another set of characters. The syntax for the replace function is: <code>replace(string1,string_to_replace,[replacement_string])</code> <code>string1</code> is the string to replace a sequence of characters with another set of characters. <code>string_to_replace</code> is the string that will be searched for in string1. <code>replacement_string</code> is optional. All occurrences of <code>string_to_replace</code> will be replaced with <code>replacement_string</code> in <code>string1</code>.</td>
</tr>
<tr>
<td><strong>to_number('12345')</strong></td>
<td><code>&lt;?xdofx:to_number('12345')&gt;</code></td>
<td>Function <code>to_number</code> 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.</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>
<tr>
<td><code>format_number</code></td>
<td><code>&lt;xdoxslt:format_number(12345, n, $_XDOLOCALE)?&gt;</code></td>
<td>Converts a number to a string and formats the number according to the locale specified in <code>$XDOLOCALE</code> and to the number of decimal positions specified in <code>n</code> using Java's default symbols. For example: ```xml<code>&lt;xdoxslt:format_number(-12345, 2, 'fr-FR')?&gt;</code> returns -12 345,00</td>
</tr>
<tr>
<td><code>format_number</code></td>
<td><code>&lt;xdoxslt:format_number(12345, n, s1, s2, $_XDOLOCALE)?&gt;</code></td>
<td>Converts a number to a string and uses the specified separators: <code>s1</code> for the thousand separator and <code>s2</code> for the decimal separator. For example: ```xml<code>&lt;xdoxslt:format_number(12345, 2, 'g', 'd', $_XDOLOCALE)?&gt;</code> returns 12g345d00</td>
</tr>
<tr>
<td><code>pat_format_number</code></td>
<td><code>&lt;xdoxslt:pat_format_number(12345, '##,##0.00', $_XDOLOCALE)?&gt;</code></td>
<td>Returns a number formatted with the specified pattern. For example: ```xml<code>&lt;xdoxslt:pat_format_number(12345, '##,##0.00', $_XDOLOCALE)?&gt;</code> returns 12,345.00</td>
</tr>
<tr>
<td><code>to_char(12345)</code></td>
<td><code>&lt;xdofx:to_char('12345')?&gt;</code></td>
<td>Use the TO_CHAR function to translate a value of NUMBER datatype to VARCHAR2 datatype.</td>
</tr>
<tr>
<td><code>to_date</code></td>
<td><code>&lt;xdofx:to_date ( char [, fmt [, 'nlsparam'] ] )</code></td>
<td>TO_DATE converts char of CHAR, VARCHAR2, NCHAR, or NVARCHAR2 datatype to a value of DATE datatype. The <code>fmt</code> is a date format specifying the format of <code>char</code>. If you omit <code>fmt</code>, then <code>char</code> must be in the default date format. If <code>fmt</code> is <code>J</code>, for Julian, then <code>char</code> must be an integer.</td>
</tr>
<tr>
<td>SQL Statement or XSL Expression</td>
<td>Usage</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>sysdate()</code></td>
<td><code>&lt;?xdofx:sysdate() ?&gt;</code></td>
<td>SYSDATE returns the current date and time. The datatype of the returned value is DATE. The function requires no arguments.</td>
</tr>
<tr>
<td><code>current_date()</code></td>
<td><code>&lt;?xdoxslt:current_date($_XDOLOCALE, $_XDOTIMEZONE) ?&gt;</code></td>
<td>Returns the current date in &quot;yyy-MM-dd&quot; format in the given locale and timezone. This function supports only the Gregorian calendar. Example: <code>&lt;?xdoxslt:current_date('ja-JP', 'Asia/Tokyo') ?&gt;</code></td>
</tr>
<tr>
<td><code>current_time()</code></td>
<td><code>&lt;?xdoxslt:current_time($_XDOLOCALE, $_XDOTIMEZONE) ?&gt;</code></td>
<td>Returns the current time in the given locale and timezone. This function supports only the Gregorian calendar. Example: <code>&lt;?xdoxslt:current_time('ja-JP', 'Asia/Tokyo') ?&gt;</code></td>
</tr>
<tr>
<td><code>minimum</code></td>
<td><code>&lt;?xdoxslt:minimum(ELEMENT_NAME) ?&gt;</code></td>
<td>Returns the minimum value of the element in the set.</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>
| date_diff                       | `<xdoxslt:date_diff('y', 'YYYY-MM-DD', 'YYYY-MM-DD', $_XDOLOCALE, $_XDOTIMEZONE)?>` | This function provides a method to get the difference between two dates in the given locale. The dates need to be in "yyyy-MM-dd" format. This function supports only the Gregorian calendar. The syntax is as follows: `<xdoxslt:date_diff('format', 'YYYY-MM-DD', 'YYYY-MM-DD', $_XDOLOCALE, $_XDOTIMEZONE)?>`  
where  
format is the time value for which the difference is to be calculated. Valid values are:  
• y - for year  
• m - for month  
• w - for week  
• d - for day  
• h - for hour  
• mi - for minute  
• s - for seconds  
• ms - for milliseconds  
Example: `<xdoxslt:date_diff('y', '2000-04-08', '2001-05-01', $_XDOLOCALE, $_XDOTIMEZONE)?>` returns 1  
Example: `<xdoxslt:date_diff('m', '2001-04-08', '2000-02-01', $_XDOLOCALE, $_XDOTIMEZONE)?>` returns |
### SQL Statement or XSL Expression Usage Description

<table>
<thead>
<tr>
<th>SQL Statement or XSL Expression</th>
<th>Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sec_diff</code></td>
<td><code>&lt;xdoxslt:sec_diff('2000-04-08T20:00:00', '2000-04-08T21:00:00', $_XDOLOCALE, $_XDOTIMEZONE)</code></td>
<td>This function provides a method to get the difference between two dates in seconds in the given locale. The dates need to be in &quot;yyyy-MM-dd'T'HH:mm:ss&quot;. This function supports only Gregorian calendar. Example: <code>&lt;xdoxslt:sec_diff('2000-04-08T20:00:00', '2000-04-08T21:00:00', $_XDOLOCALE, $_XDOTIMEZONE)</code> returns 3600</td>
</tr>
<tr>
<td><code>get_day</code></td>
<td><code>&lt;xdoxslt:get_day('2000-04-08', $_XDOLOCALE)</code></td>
<td>This function provides a method to get the day value of a date in &quot;yyyy-MM-dd&quot; format in the given locale. This function supports only the Gregorian calendar. Example: <code>&lt;xdoxslt:get_day('2000-04-08', $_XDOLOCALE)</code> returns 8</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>
<tr>
<td>get_month</td>
<td></td>
<td>This function provides a method to get the month value of a date in &quot;yyyy-MM-dd&quot; format in the given locale. This function supports only the Gregorian calendar.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;?xdoxslt:get_month('2000-04-08', $_XDOLOCALE)?&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>returns</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>get_year</td>
<td></td>
<td>This function provides a method to get the year value of a date in &quot;yyyy-MM-dd&quot; format in the given locale. This function supports only the Gregorian calendar.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;?xdoxslt:get_year('2000-04-08', $_XDOLOCALE)?&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>returns</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2000</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>
| month_name                      | <?xdoxslt:month_name(1, 0, $_XDOLOCALE)?>                             | This function provides a method to get the name of the month in the given locale. This function supports only the Gregorian calendar. The syntax for this function is:  

<?xdoxslt:month_name(month, [abbreviate?], $_XDOLOCALE)?>

where

month is the numeric value of the month (January = 1) and [abbreviate?] is the value 0 for do not abbreviate or 1 for abbreviate. Example:

<?xdoxslt:month_name(12, 1, 'fr-FR')?> returns dec. Example:

<?xdoxslt:month_name(1, 0, $_XDOLOCALE)?> returns January |
| maximum                         | <?xdoxslt:maximum(ELEMENT_NAME)?>                                   | Returns the maximum value of the element in the set. |
| abs                             | <?xdoxslt:abs(-123.45)?>                                            | Returns the absolute value of the number entered. Example:

<?xdoxslt:abs(-123.45)?> Returns: 123.45 |
<table>
<thead>
<tr>
<th>SQL Statement or XSL Expression</th>
<th>Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chr</td>
<td>&lt;?xdofx:chr(n)?&gt;</td>
<td>CHR returns the character having the binary equivalent to n in either the database character set or the national character set.</td>
</tr>
<tr>
<td>ceil</td>
<td>&lt;?xdofx:ceil(n)?&gt;</td>
<td>CEIL returns smallest integer greater than or equal to n.</td>
</tr>
<tr>
<td>floor</td>
<td>&lt;?xdofx:floor(n)?&gt;</td>
<td>FLOOR returns largest integer equal to or less than n.</td>
</tr>
<tr>
<td>round</td>
<td>&lt;?xdofx:round ( number [, integer ] )?&gt;</td>
<td>ROUND returns number rounded to integer places right of the decimal point. If integer is omitted, then number is rounded to 0 places. integer can be negative to round off digits left of the decimal point. integer must be an integer. Example: &lt;?xdofx:round (2.777)?&gt; returns 3 Example: &lt;?xdofx:round (2.777, 2)?&gt; returns 2.78</td>
</tr>
</tbody>
</table>
### Extended Function Support in RTF Templates

<table>
<thead>
<tr>
<th>SQL Statement or XSL Expression</th>
<th>Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>round</strong>&lt;br&gt;(XSLT function)</td>
<td>&lt;?xdoxslt:round ( number [, integer ] )?&gt;</td>
<td>ROUND returns number rounded to integer places right of the decimal point. If integer is omitted, then number is rounded to 0 places. integer can be negative to round off digits left of the decimal point. integer must be an integer. Example: &lt;?xdoxslt:round (2.777)?&gt; returns 3. Example: &lt;?xdoxslt:round (2.777, 2)?&gt; returns 2.78</td>
</tr>
<tr>
<td><strong>lower</strong></td>
<td>&lt;?xdofx:lower (char)?&gt;</td>
<td>LOWER returns char, with all letters lowercase. char can be any of the datatypes CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB. The return value is the same datatype as char.</td>
</tr>
<tr>
<td><strong>upper</strong></td>
<td>&lt;?xdofx:upper(char)?&gt;</td>
<td>UPPER returns char, with all letters uppercase. char can be any of the datatypes CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB. The return value is the same datatype as char.</td>
</tr>
<tr>
<td><strong>length</strong></td>
<td>&lt;?xdofx:length(char)?&gt;</td>
<td>The &quot;length&quot; function returns the length of char. LENGTH calculates length using characters as defined by the input character set.</td>
</tr>
<tr>
<td><strong>greatest</strong></td>
<td>&lt;?xdofx:greatest ( expr [, expr]... )?&gt;</td>
<td>GREATEST returns the greatest of the list of exprs. All exprs after the first are implicitly converted to the datatype of the first expr before the comparison.</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>
<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>
<tr>
<td>next_element</td>
<td><code>&lt;xdoxslt:next_element(current-group(),.,'&lt;element-name&gt;')?&gt;</code></td>
<td>Method to get the next element in the current group. Will return the element that occurs after the element named. For example: <code>&lt;xdoxslt:next_element(current-group(),.,'employee')?&gt;</code> will return the element that occurs in the current group after &quot;employee&quot;.</td>
</tr>
<tr>
<td>prev_element</td>
<td><code>&lt;xdoxslt:prev_element(current-group(),..,'&lt;element-name&gt;')?&gt;</code></td>
<td>Method to get the previous element in the current group. Will return the element that occurs before the element named. For example: <code>&lt;xdoxslt:prev_element(current-group(),..,'employee')?&gt;</code> will return the element that occurs in the current group before &quot;employee&quot;.</td>
</tr>
</tbody>
</table>
| set_array                       | `<xdoxslt:set_array($_XDOCTX, '<name of hash table>', n, '<value>')?>` | Sets a value in a hash table. Syntax is `<xdoxslt:set_array($_XDOCTX, '<name of hash table>', n, '<value>')?>` where 

$_XDOCTX is required to set the context, 

`<name of hash table>` is the name you supply for your table 

`n` is the index of the hash table 

`<value>` is the value to set in the hash table. 

For example: `<xdoxslt:set_array($_XDOCTX, 'Employee', 2, 'Jones')?>` See get_array below. |
Extended Function Support in RTF Templates

### SQL Statement or XSL Expression

<table>
<thead>
<tr>
<th>SQL Statement or XSL Expression</th>
<th>Usage</th>
<th>Description</th>
</tr>
</thead>
</table>
| get_array                      | `<?xdoxslt:get_array($_XDOCTX, '<name of hash table>', n)?>` | Returns the value at the specified index of the hash table. Syntax is `<?xdoxslt:get_array($_XDOCTX, '<name of hash table>', n)?>` where 
   - `$._XDOCTX` is required to set the context, 
   - `<name of hash table>` is the name you supplied for your table in set_array 
   - `n` is the index value of the element you want returned. For example, used in conjunction with the set_array example above, `<?xdoxslt:get_array($_XDOCTX, 'Employee', 2)?>` returns Jones |

The following table shows supported combination functions:

<table>
<thead>
<tr>
<th>SQL Statement</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>(2+3/4-6*7)/8</code></td>
<td><code>&lt;?xdofx:(2+3/4-6*7)/8?&gt;</code></td>
</tr>
<tr>
<td><code>lpad(substr('1234567890',5,3),10,'^')</code></td>
<td><code>&lt;?xdofx:lpad(substr('1234567890',5,3),10,'^')?&gt;</code></td>
</tr>
<tr>
<td>`decode('a','b','c','d','e','1')</td>
<td></td>
</tr>
</tbody>
</table>

**XSL Equivalents**

The following table lists the BI Publisher simplified syntax with the XSL equivalents.
<table>
<thead>
<tr>
<th>Supported XSL Elements</th>
<th>Description</th>
<th>BI Publisher Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;xsl:value-of select=&quot;name&quot;&gt;</code></td>
<td>Placeholder syntax</td>
<td><code>&lt;?name?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xsl:apply-templates select=&quot;name&quot;&gt;</code></td>
<td>Applies a template rule to the current element's child nodes.</td>
<td><code>&lt;?apply:name?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xsl:copy-of select=&quot;name&quot;&gt;</code></td>
<td>Creates a copy of the current node.</td>
<td><code>&lt;?copy-of:name?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xsl:call-template name=&quot;name&quot;&gt;</code></td>
<td>Calls a named template to be inserted into/applied to the current template.</td>
<td><code>&lt;?call:name?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xsl:sort select=&quot;name&quot;&gt;</code></td>
<td>Sorts a group of data based on an element in the dataset.</td>
<td><code>&lt;?sort:name?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xsl:for-each select=&quot;name&quot;&gt;</code></td>
<td>Loops through the rows of data of a group, used to generate tabular output.</td>
<td><code>&lt;?for-each:name?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xsl:choose&gt;</code></td>
<td>Used in conjunction with <code>when</code> and <code>otherwise</code> to express multiple conditional tests.</td>
<td><code>&lt;?choose?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xsl:when test=&quot;exp&quot;&gt;</code></td>
<td>Used in conjunction with <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>
</tbody>
</table>
### Supported XSL Elements

<table>
<thead>
<tr>
<th>Supported XSL Elements</th>
<th>Description</th>
<th>BI Publisher Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;xsl:stylesheet xmlns:x=&quot;url&quot;&gt;</code></td>
<td>Define the root element of a</td>
<td><code>&lt;?namespace:x=url?&gt;</code></td>
</tr>
<tr>
<td></td>
<td>stylesheet</td>
<td></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 BI Publisher RTF templates. The BI Publisher syntax can be used with either RTF template method.

The full list of FO elements supported by BI Publisher can be found in the Appendix: Supported XSL-FO Elements, page A-1.

<table>
<thead>
<tr>
<th>FO Element</th>
<th>BI Publisher Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;fo:page-number-citation ref-id=&quot;id&quot;&gt;</code></td>
<td><code>&lt;?fo:page-number-citation:id?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;fo:page-number&gt;</code></td>
<td><code>&lt;?fo:page-number?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;fo:ANY NAME WITHOUT ATTRIBUTE&gt;</code></td>
<td><code>&lt;?fo:ANY NAME WITHOUT ATTRIBUTE?&gt;</code></td>
</tr>
</tbody>
</table>
This chapter covers the following topics:

- Template Translations
- Report File Translations

**Template Translations**

There are two options for adding translated templates to your report definition:

- Create a separate RTF template that is translated (a localized template)
- Generate an XLIFF file from the original template (at runtime the original template is applied for the layout and the XLIFF file is applied for the translation)

Use the first option if the translated template requires a different layout from the original template.

If you only require translation of the text strings of the template layout, use the XLIFF option.

**Important:** Regardless of which option you choose, you must name your translated templates according to the naming standard for BI Publisher to recognize it at runtime. See Naming Standards for Translated Files, page 9-6.

The following diagrams illustrate the translation concepts
**Localized Template Option**

Use this option when you want a specific translation to have a different layout.

```
Data
  xml
  EmployeeReport.xml
  EmployeeTemplate_fr_FR.rtf

Localized (or translated) Template

BI Publisher

Employés

Output report in French
```

**XLIFF File Option**

Use this option when you want to use the same layout and apply specific translations.

```
Data
  xml
  EmployeeReport.xml

Base Template (in English)
  Employees
  EmployeeTemplate.rtf
  EmployeeTemplate_fr_FR.xlf

Translated Text Strings
  xlf

BI Publisher

Axial John
Gomez
Smith
Alex

Employees

Output report in French
```
Using the XLIFF Option

To generate an XLIFF file from an RTF template:

1. Open your template in Microsoft Word with the Template Builder for Word installed.

2. From the Template Builder menu, select Tools > Translations > Extract Text.
   BI Publisher extracts the translatable strings from the template and exports them to an XLIFF (xlf) file.

3. Save the file to the desired location.

This XLIFF file can then be sent to a translation provider, or using a text editor, you can enter the translation for each string. See Structure of the XLIFF File, page 9-4 for instructions on how to edit the XLIFF file.

Note: XLIFF is the XML Localization Interchange File Format. It is the standard format used by localization providers. For more information about the XLIFF specification, see http://www.oasis-open.org/committees/xliff/documents/xliff-specification.htm

A "translatable string" is any text in the template that is intended for display in the published report, such as table headers and field labels. Text supplied at runtime from
the data is not translatable, nor is any text that you supply in the Microsoft Word form fields. You can translate the template XLIFF file into as many languages as desired and then associate these translations to the original template. See Uploading Translated Files, page 9-6.

Structure of the XLIFF File

The generated XLIFF file has the following structure:

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

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

```xml
```

source-language and target-language attributes

The `<file>` element includes the attributes `<source-language>` and `<target-language>`. The valid value for source-language and target-language is a combination of the language code and country code as follows:

- the two-letter ISO 639 language code
- the two-letter ISO 3166 country code
Note: For more information on the International Organization for Standardization (ISO) and the code lists, see International Organization for Standardization [http://www.iso.org/iso/en/ISOOnline.frontpage].

For example, the value for English-United States is "en-US". This combination is also referred to as a locale.

When you edit the exported XLIFF file you must change the target-language attribute to the appropriate locale value of your target language. The following table shows examples of source-language and target-language attribute values appropriate for the given translations:

<table>
<thead>
<tr>
<th>Translation (Language/Territory)</th>
<th>source-language value</th>
<th>target-language value</th>
</tr>
</thead>
<tbody>
<tr>
<td>From English/US</td>
<td>en-US</td>
<td>en-CA</td>
</tr>
<tr>
<td>To English/Canada</td>
<td></td>
<td></td>
</tr>
<tr>
<td>From English/US</td>
<td>en-US</td>
<td>zh-CN</td>
</tr>
<tr>
<td>To Chinese/China</td>
<td></td>
<td></td>
</tr>
<tr>
<td>From Japanese/Japan</td>
<td>ja-JP</td>
<td>fr-FR</td>
</tr>
<tr>
<td>To French/France</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Embedded Data Fields

Some templates contain placeholders for data fields embedded in the text display strings of the report. For example, the title of the sample report is

**Italian Purchase VAT Register - (year)**

where (year) is a placeholder in the RTF template that will be populated at runtime by data from an XML element. These fields are not translatable, because the value comes from the data at runtime.

To identify embedded data fields, the following token is used in the XLIFF file:

`[&n]`

where n represents the numbered occurrence of a data field in the template.

For example, in the preceding XLIFF sample, the first translatable string is

```
<source>Italian Purchase VAT Register - [&1]<source>
```

**Warning:** Do not edit or delete the embedded data field tokens or you
will affect the merging of the XML data with the template.

**<source> and <target> Elements**

Each `<source>` element contains a translatable string from the template in the source language of the template. For example,

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

When you initially export the XLIFF file for translation, the source and target elements are all identical. To create the translation for this template, enter the appropriate translation for each source element string in its corresponding `<target>` element.

Therefore if you were translating the sample template into German, you would enter the following for the Total string:

```
<source>Total</source>
<target>Gesamtbetrag</target>
```

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

```
<?xml version='1.0' encoding='utf-8'?>
<xliif version='1.0'>
 <file source-language='en-US' target-language='zh-CN' datatype='XLIFF'>
   <header/>
   <body>
     <trans-unit id="6f7824d" msgid="4000" msgstr="90" size-un
 <source>Italian Purchase VAT Register - [(amp;1)]</source>
 <target>意大利购买VAT 记录 - [(amp;1)]</target>
 <note>Text located: header/table, token &amp;amp;1:anonymous pisa</note>
 </trans-unit>
 <trans-unit id="4d2eb24" msgid="4000" msgstr="15" size-un
 <source>Total</source>
 <target>共計</target>
 <note>Text located: body/table</note>
 </trans-unit>
 <trans-unit id="6c2cc17e" msgid="4000" msgstr="37" size-un
 <source>Non-Recoverable</source>
 <target>非恢复性</target>
 <note>Text located: body/table</note>
 </trans-unit>
</body>
</file>
```

**Naming Standards for Translated Files**

Your translated XLIFF and RTF files must be named according to the following standard:

TemplateName_<language code>_<TERRITORY CODE>.xlf or .rtf

or

TemplateName_<language code>.xlf or .rtf
where TemplateName is the original template name

*language code* is the two-letter ISO language code (in lower case)

*TERRITORY CODE* is the two-letter ISO country code (must be in upper case)

For example, if your original template is named EmployeeTemplate and you are uploading a translation for Japanese-Japan, name the file:

EmployeeTemplate Ja_JP.xlf

**Uploading Translated Files**

In the report Editor, select the **Layouts** page to upload the translated XLIFF files. See Define Layouts, page 4-13.

**Locale Selection Logic**

BI Publisher applies a translation based on the user's selected Report Locale. BI Publisher will first try to match an RTF template named for the locale, then an XLIFF file named for the locale. If an exact match on language-territory is not found, BI Publisher will try to match on language only.

For example, if you have a report for which the base template is called EmployeeTemplate.rtf and the locale selected is French (France), BI Publisher will select the translation to apply according to the following hierarchy:

EmployeeTemplate_fr_FR.rtf
EmployeeTemplate_fr_FR.xlf
EmployeeTemplate_fr.rtf
EmployeeTemplate_fr.xlf
EmployeeTemplate.rtf

Note that with the same set of translations, if the locale selected is French (Switzerland), the EmployeeTemplate_fr.rtf would be applied. Now if the available translations were limited to the following set:

EmployeeTemplate_fr_FR.rtf
EmployeeTemplate_fr_FR.xlf
EmployeeTemplate.rtf

and the locale selected is French (Switzerland), then the EmployeeTemplate.rtf will be applied. Even though there is a language match, BI Publisher will not match the different locales.

Therefore, if you want to ensure that a French language translation is used when French is the selected language, regardless of the selected locale, then you must include either an rtf or xlf file named for the language only (that is, EmployeeTemplate_fr.rtf or EmployeeTemplate_fr.xlf).
Report File Translations

You can add translated report description files so that your users can view the report description and any parameter labels in the language they selected for their UI preference. Upload translated report description files to the same location as the translated template files. Note that the translated report description files follow a naming standard that is slightly different than the translated template file standard.

For information on setting the UI language preference, see Setting Preferences, page 2-2.

To add a report file translation:
1. In the report Editor, select Generate XLIFF.
2. Save the .xlf file to a local directory.
3. Send the file to a localization provider, or add the translated text (see Structure of the XLIFF File, page 9-4 for information on editing the XLIFF file).
4. Name the translated report file according to the following standard for all languages except Chinese and Portuguese (Brazil):
   ReportName_<language_code>.xlf
   Where ReportName is the report file name and
   language_code is the two-letter ISO language code (in lower case).
   Important: Except for the three locales noted below, do not include the territory code in the file name.

   For Chinese (China), Chinese (Taiwan), and Portuguese (Brazil) you must use the language code and territory code in the translated file name as follows:
   ReportName_zh_CN.xlf
   ReportName_zh_TW.xlf
   ReportName_pt_BR.xlf

5. In the report Editor, select the Layouts page to upload the translated XLIFF files. See Define Layouts, page 4-13.
Overview

To create a PDF template, take any existing PDF document and apply the BI 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 Predefined PDF Form, page 10-17.

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**

BI 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, BI 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.

### Purchase Order

<table>
<thead>
<tr>
<th>E-Business Suite</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>VENDOR:</td>
<td></td>
</tr>
<tr>
<td>CUSTOMER ACCOUNT NO.</td>
<td>VENDOR NO.</td>
</tr>
<tr>
<td>PAYMENT TERMS</td>
<td>SHIP VIA</td>
</tr>
<tr>
<td>FREIGHT TERMS</td>
<td>REQUESTED/DELIVERED</td>
</tr>
<tr>
<td>FOB</td>
<td>QUANTITY</td>
</tr>
<tr>
<td>TOTAL</td>
<td></td>
</tr>
</tbody>
</table>

Oracle E-Business Suite
Creating a PDF Template

10-3

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

```xml
<?xml version="1.0"?>
<POXPRPOP2>
  <G_HEADERS>
    <POH_PO_NUM>1190-1</POH_PO_NUM>
    <POH_REVISION_NUM>0</POH_REVISION_NUM>
    <POH_SHIP_ADDRESS_LINE1>3455 108th Avenue</POH_SHIP_ADDRESS_LINE1>
    <POH_SHIP_ADDRESS_LINE2></POH_SHIP_ADDRESS_LINE2>
    <POH_SHIP_ADDRESS_LINE3></POH_SHIP_ADDRESS_LINE3>
    <POH_SHIP_ADR_INFO>Seattle, WA 98101</POH_SHIP_ADR_INFO>
    <POH_SHIP_COUNTRY>United States</POH_SHIP_COUNTRY>
    <POH_VENDOR_NAME>Allied Manufacturing</POH_VENDOR_NAME>
    <POH_VENDOR_ADDRESS_LINE1>1145 Brokaw Road</POH_VENDOR_ADDRESS_LINE1>
    <POH_VENDOR_ADDRESS_LINE2></POH_VENDOR_ADDRESS_LINE2>
    <POH_VENDOR_ADR_INFO>San Jose, CA 95034</POH_VENDOR_ADR_INFO>
    <POH_VENDOR_COUNTRY>United States</POH_VENDOR_COUNTRY>
    <POH_BILL_ADDRESS_LINE1>90 Fifth Avenue</POH_BILL_ADDRESS_LINE1>
    <POH_BILL_ADDRESS_LINE2></POH_BILL_ADDRESS_LINE2>
    <POH_BILL_ADR_INFO>New York, NY 10022-3422</POH_BILL_ADR_INFO>
    <POH_BILL_COUNTRY>United States</POH_BILL_COUNTRY>
    <POH_BUYER>Smith, J</POH_BUYER>
    <POH_PAYMENT_TERMS>45 Net (terms date + 45)</POH_PAYMENT_TERMS>
    <POH_SHIP_VIA>UPS</POH_SHIP_VIA>
    <POH_FREIGHT_TERMS>Due</POH_FREIGHT_TERMS>
    <POH_CURRENCY_CODE>USD</POH_CURRENCY_CODE>
    <POH_CURRENCY_CONVERSION_RATE></POH_CURRENCY_CONVERSION_RATE>
  </G_HEADERS>
  <LIST_G_LINES>
    <G_LINES>
      <POL_LINE_NUM>1</POL_LINE_NUM>
      <POL_VENDOR_PRODUCT_NUM></POL_VENDOR_PRODUCT_NUM>
      <POL_ITEM_DESCRIPTION>PCMCIA II Card Holder</POL_ITEM_DESCRIPTION>
      <POL_QUANTITY_TO_PRINT></POL_QUANTITY_TO_PRINT>
      <POL_PRICE_TO_PRINT>15</POL_PRICE_TO_PRINT>
      <C_FLEX_ITEM>CM16374</C_FLEX_ITEM>
      <C_FLEX_ITEM_DISP>CM16374</C_FLEX_ITEM_DISP>
      <PLL_QUANTITY_ORDERED>7500</PLL_QUANTITY_ORDERED>
      <C_AMOUNT_PLL>112500</C_AMOUNT_PLL>
      <C_AMOUNT_PLL_DISP>112,500.00</C_AMOUNT_PLL_DISP>
    </G_LINES>
  </LIST_G_LINES>
</POXPRPOP2>
```

Adding Markup to the Template Layout

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

The process of associating the XML data to the PDF template is the same as the process for the RTF template. See: Associating the XML Data to the Template Layout: Associating the XML data to the template layout, page 7-3.

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

Creating a Placeholder

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

**Note:** If you are using Adobe Acrobat 5.0, the **Form Tool** is available from the standard toolbar. If you are using Adobe Acrobat 6.0 or later, display the **Forms Toolbar** from the Tools menu by selecting Tools > Advanced Editing > Forms > Show Forms Toolbar.

Naming the Placeholder

The name of the placeholder must match the XML source field name.

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**

BI 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 10-16.

- **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:

<?rep_field="BODY_START"?>

5. Define the end of the group area by inserting a form field at the end of the area the that is to contain the group.

6. In the Name field of the Field Properties window, enter any 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:

<?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 10-15.

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):

<?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

  <?rep_field="T1_G1"?>

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

  <?rep_field="T1_G2"?>

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

  <?rep_field="T1_G3"?>

The following graphic shows the entries for the Short Description/Tooltip field:
3. (Optional) Align your fields. To ensure proper alignment of a row of fields, it is recommended that you use Adobe Acrobat's alignment feature.

**Adding Page Numbers**

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 10-4).

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:
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>DEPTNO</td>
<td>EMPNO</td>
<td>ENAME</td>
<td>JOB</td>
<td>SAL</td>
</tr>
<tr>
<td>SUMSALPERDEPTNO</td>
<td></td>
<td></td>
<td></td>
<td></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:

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

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

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

The sample report with data is shown in the following figure:
### Department Salary Summary

<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>10</td>
<td>7782</td>
<td>CLARK</td>
<td>MANAGER</td>
<td>2450</td>
</tr>
<tr>
<td></td>
<td>7839</td>
<td>KING</td>
<td>PRESIDENT</td>
<td>5000</td>
</tr>
<tr>
<td></td>
<td>125</td>
<td>KANG</td>
<td>CLERK</td>
<td>2000</td>
</tr>
<tr>
<td></td>
<td>7934</td>
<td>MILLER</td>
<td>CLERK</td>
<td>1300</td>
</tr>
<tr>
<td></td>
<td>123</td>
<td>MARY</td>
<td>CLERK</td>
<td>400</td>
</tr>
<tr>
<td></td>
<td>124</td>
<td>TOM</td>
<td>CLERK</td>
<td>3000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9150</td>
</tr>
</tbody>
</table>

<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>20</td>
<td>7369</td>
<td>SMITH</td>
<td>CLERK</td>
<td>800</td>
</tr>
<tr>
<td></td>
<td>7876</td>
<td>ADAMS</td>
<td>CLERK</td>
<td>1100</td>
</tr>
<tr>
<td></td>
<td>7902</td>
<td>FORD</td>
<td>ANALYST</td>
<td>3000</td>
</tr>
<tr>
<td></td>
<td>7788</td>
<td>SCOTT</td>
<td>ANALYST</td>
<td>3000</td>
</tr>
<tr>
<td></td>
<td>7566</td>
<td>JONES</td>
<td>MANAGER</td>
<td>2975</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10875</td>
</tr>
</tbody>
</table>

The page breaks after each department.

### 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 Example**

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, BI 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, BI 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 BI Publisher Configuration File, *Oracle Business Intelligence Publisher Administrator’s 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 Predefined PDF Form
There are many PDF forms available online that you may want to use as templates for your report data. For example, government forms that your company is required to submit. You can use these downloaded PDF files as your report templates, supplying the XML data at runtime to fill the report out.

Some of these forms already have form fields defined, some do not. If the form already has fields defined, you can either use BI Publisher's Mapping tool (see Adding a Predefined Form as a Template, page 4-25) or name your data fields to match the form field names (see Using a Predefined Form as a Template by Matching Form Fields, page 10-18). If the form fields are not already defined in the downloaded PDF, you must create them. See Adding Markup to the Template Layout, page 10-3 for instructions on inserting the form field placeholders.
Using a Predefined PDF Form as a Template by Matching the Form Fields

1. Download or import the PDF file to your local system.

2. Open the file in Adobe Acrobat.

3. Select the Text Field Tool (Acrobat 6.0 users) or the Form Tool (Acrobat 5.0 users). This will highlight text fields that have already been defined.

The following figure shows a sample W-4 PDF form after selecting the Text Field Tool to highlight the text fields (in Acrobat 6.0).

To map the existing form fields to the data from your incoming XML file, you must rename the fields to match the element names in your XML file.

4. Open the text form field Properties dialog by either double-clicking the field, or by selecting the field then selecting Properties from the right-mouse menu.

5. In the Name field, enter the element name from your input XML file.

6. Repeat for all fields that you want populated by your data file.

Adding or Designating a Field for a Digital Signature

Oracle BI Publisher supports digital signatures on PDF output documents. Digital signatures enable you to verify the authenticity of the documents you send and receive. Oracle BI Publisher can access your digital ID file from a central, secure location and at
runtime sign the PDF output with the digital ID. The digital signature verifies the
signer’s identity and ensures that the document has not been altered after it was signed.
Implementing digital signature requires several tasks across the BI Publisher product.
This topic describes how to add a new field or configure an existing field in your PDF
template for the digital signature. For more information and a description of the other
required tasks and options, see Implementing a Digital Signature, Oracle Business
Intelligence Publisher Administrator’s and Developer’s Guide.

About Signature Field Options
For PDF templates you have the following options for designating a digital signature
field for your output report:

• Add a signature field to the PDF template.
  Use this option if you want the digital signature to appear in a specific field and
  your PDF template does not already include a signature field. See Adding a
  Signature Field, page 10-19.

• Use an existing signature field in the PDF template.
  Use this option if your PDF template already includes a signature field that you
  want to use. To designate an existing field for the digital signature, define the field
  in the Runtime Configuration page. See Configuring the Report to Insert the Digital
  Signature at Runtime, page 10-21.

• Designate the position of the digital signature on the output report by setting x and
  y coordinates.
  Use this option if you prefer to designate the x and y coordinates for the placement
  of the digital signature, rather than use a signature field. You set the position using
  runtime properties. For information on setting these properties, see PDF Digital
  Signature Properties, page 13-6.

All three options require setting configuration properties for the report in BI Publisher’s
Runtime Configuration page after you have uploaded the template.

Adding a Signature Field
To add a signature field:

1. Open the template in Adobe Acrobat Professional.

2. Select Sign from the toolbar and then select Create a Blank Signature Field. The
   following figure displays this selection from the Adobe Acrobat Professional
   toolbar.
3. Draw the signature field in the desired location on the template. When you are finished drawing (that is, when you release the mouse button), the Digital Signature Properties dialog will display.

4. Assign a name to your signature field. The following figure shows an inserted digital signature field called "My_Signature."
5. Save your template.


Configuring the Report to Insert the Digital Signature at Runtime

After you have uploaded your report to the report definition (see Adding a Layout, page 4-14) you must enable digital signature and specify the signature field on the Runtime Configuration page.

1. Select the Configure link for the report.

2. On the Runtime Configuration page, set Enable Digital Signature to True.

3. Enter the field name from the PDF template for the property Existing signature field name.

The following figure shows the "My_Signature" field name entered into the properties field.
<table>
<thead>
<tr>
<th>Properties</th>
<th>Report Value</th>
<th>Server Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bursting</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enable multithreading</td>
<td></td>
<td>False</td>
</tr>
<tr>
<td>Thread count</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>PDF Output</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compress PDF output</td>
<td></td>
<td>True</td>
</tr>
<tr>
<td>Hide PDF viewer's menu bar</td>
<td></td>
<td>False</td>
</tr>
<tr>
<td>PDF Digital Signature</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enable Digital Signature</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>Existing signature field name</td>
<td>My_Signature</td>
<td></td>
</tr>
<tr>
<td>Signature field location</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Signature field X coordinate</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Signature field Y coordinate</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Signature field width</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Signature field height</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTF Output</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Creating Flash Templates

Introduction

BI Publisher's support for Flash templates enables you to develop Adobe Flex templates that can be applied to BI Publisher reports to generate interactive Flash output documents.

**Note:** Adobe Flex is an open-source technology for building interactive cross-platform applications. Flex applications can be delivered using Adobe Flash Player. For more information see the Flex Web site at http://www.flex.org.

BI Publisher's integration with Flex enables you to build Flex templates, test them on your desktop, and deploy them to the BI Publisher server to generate Flash output. Users are then able to run the reports from the BI Publisher user interface or schedule them for delivery to report consumers.

This chapter will describe how to set up a Flex template with a BI Publisher "flat" data source (that is, there is no hierarchy in the XML data) and how to include simpler objects such as tables and charts. For more information about interactivity, connectivity between components and more advanced topics, refer to Adobe's Flex documentation.

Prerequisites for Building and Viewing Flash Templates

Following are the prerequisites for building and viewing Flash templates:

- For viewing output:
  - To view the report output from the Flash Template, you must have Adobe Flash Player 9 installed on your computer. If viewing reports over the BI Publisher user interface, your Web browser must also support the Adobe Flash Player 9 plug-in.
• For building templates:
  • The FlexBuilder IDE from Adobe

Oracle BI Publisher is currently certified with version 2.0.1. The tool can be downloaded and purchased from the Adobe Web site at http://www.adobe.com/products/flex/.

Note that the charting functionality requires an additional license fee.

• A report data model set up in BI Publisher that generates flat XML. For information on setting up your data model, see Defining the Data Model, page 4-6.

Building a Flash Template

This section describes how to build a Flash template and includes the following topics:

• Adding the Data Source

• Creating the Layout

• Data Binding

Adding the Data Source

To add the data source:

1. Generate a sample data file from your report data model:

   From the BI Publisher Report Editor or from the Reports page, select View. If no layouts are defined for your report, then the output type will default to XML. Otherwise, choose Data for the output type. Select Export and save the results as an XML file to a local directory.

   This example is based on the following data:

   ```xml
   <ROWSET>
   <ROW>
   <NAME>Neena Kochhar</NAME>
   <FIRST_NAME>Neena</FIRST_NAME>
   <LAST_NAME>Kochhar</LAST_NAME>
   <SALARY>17000</SALARY>
   <ANNUAL_SALARY>204000</ANNUAL_SALARY>
   <FED_WITHHELD>57120</FED_WITHHELD>
   <JOB_TITLE>Administration Vice President</JOB_TITLE>
   <DEPARTMENT_NAME>Executive</DEPARTMENT_NAME>
   <MANAGER>Steven King</MANAGER>
   </ROW>
   <ROW>
   ...
   </ROWSET>
   ```
This data is generated from the following simple query-based report:

```
select
    e.first_name || ' ' || e.last_name name,
    e.first_name,
    e.last_name,
    e.salary,
    e.salary*12 ANNUAL_SALARY,
    e.salary*12*0.28 FED_WITHHELD,
    j.job_title,
    d.department_name,
    m.first_name || ' ' || m.last_name manager
from employees e,
    employees m,
    departments d,
    jobs j
where e.department_id = d.department_id
    and j.job_id = e.job_id
    and e.manager_id = m.employee_id
```

2. Open the Flex IDE and create a new Flex Project; select the "Basic" data access method, as shown in the following example.

In the next dialog, give the project a name as shown in the following example. The name you use here will be assigned to the template file name you are going to create.
Click Finish.

The IDE creates the Flex template definition file, which is an MXML file. An MXML file is an XML format. Following is a sample:

```xml
<?xml version="1.0" encoding="utf-8"?>
    layout="absolute">
</mx:Application>
```

You can now update it manually or by using the visual builder.

3. Connect the XML you downloaded from your report data model:

To connect the data, use the XML data services that Flex supports and embed the sample data into the MXML file.

The sample MXML file with the connected data is shown. See the following section for a description of the file components.
<?xml version="1.0" encoding="utf-8"?>
layout="absolute">
<mx:Script>
<![[CDATA[
 [Bindable]
 public var dataXML:XML =  
 <ROWSET>
 <ROW>
  <NAME>Neena Kochhar</NAME>
  <FIRST_NAME>Neena</FIRST_NAME>
  <LAST_NAME>Kochhar</LAST_NAME>
  <SALARY>17000</SALARY>  
  <ANNUAL_SALARY>204000</ANNUAL_SALARY>  
  <FED_WITHHELD>57120</FED_WITHHELD>  
  <JOB_TITLE>Administration Vice President</JOB_TITLE>  
  <DEPARTMENT_NAME>Executive</DEPARTMENT_NAME> 
  <MANAGER>Steven King</MANAGER> 
 </ROW>
 <ROW> 
 ...  
 </ROWSET>;
]]> 
</mx:Script>
</mx:Application>

The XML portion should look familiar as the data you downloaded. The additional components to note are:

- <mx:Script> — This denotes the start of the template scripting code. There is also a closing </mx:Script> statement.

- [Bindable] — This denotes that the following variable is bindable to a layout component.

- public var dataXML:XML — This is the data variable declaration:
  - public — The value of the variable is available to the whole template.
  - var — Declares there is a variable in the report.
  - dataXML — The name of the variable. Note this is a compulsory name. You must use this name to use the template with BI Publisher.
  - :XML — Declares that the variable is an XML type.

- ; — Notice the semicolon after the end of the XML data you provided.

At runtime the BI Publisher server will generate the runtime data from the report and inject it into the Flex template replacing the sample data held within the dataXML variable. This feature allows the Flex report to be distributed to users without needing to connect to the server.
Creating the Layout

The Flex IDE creates a default canvas for you to drop objects onto. You can modify the canvas as required to suit your report.

**Important:** If you intend to embed the Flash output in a PDF document, you must set the Width and Height of the template in the Size region of the Layout properties. Even if you wish to accept the default size, you must explicitly enter values in these fields.

Create the layout by adding report objects to the layout palette. This example uses the Flex Design tab to add the objects to the layout. Click the Design tab to see the available objects in the Component Navigator pane.

The following figure shows an example of the available objects in the Component Navigator pane:

These objects can be dragged and dropped to the design palette.

1. Start by dragging a Panel object from under the Layout node to the design palette. Notice as you drag the panel around the edge of the palette, the guidelines are displayed in blue. Use these guides to aid you in aligning objects.

2. Drop the panel onto the top left hand corner of the palette.
3. Now drag the bottom right edge of the panel across to the right hand side of the palette.

4. Then drag it down to about half the height of the palette. Alternatively, use the property palette on the right hand side to set the size of the panel.

5. Now select a Datagrid object. This is the object to render the data in a tabular format. Drop it onto the panel you created in Step 1. The Datagrid is now a child of the panel; you can resize it as needed. The end result is shown in the following figure:

![Datagrid](image)

By default three columns are generated. In the next section, Binding the Layout Objects to the Data Source, page 11-8, you will override the default in the MXML code.

**Adding a Chart**

If you have purchased the charting option you can add charts to your layout.

1. First make some room for the chart in your layout. Highlight the Datagrid object and pull the top edge down to about half the height of the hosting panel.

2. For this example, select and drag a Column Chart from the design palette and drop it onto the hosting panel. Use the guidelines to align it.

3. Once you drop it, notice that the default size overlaps the Datagrid and that the
chart legend is in the top left-hand corner. Resize the chart and move the legend to the left to look similar to the following figure:

![Sample Chart with Legend](chart.png)

This is a sample chart. You will bind it to the data in the next section.

## Binding the Layout Objects to the Data Source

Now that the layout is complete, bind the layout objects to the data source. Flex offers some help through the property palette of the objects to define the binding, but not enough to complete the task. Therefore you must update the MXML directly using the **Source** editor.

### Binding the DataGrid

To bind the DataGrid:

1. Start by highlighting the DataGrid in the design palette, and then click the **Source** tab to display the MXML source. You will see that the first line of the DataGrid code has been highlighted for you. This is a useful feature if you have built complex Flex templates and need to locate the code easily.

   The DataGrid code is as follows:

   ```xml
   <mx:DataGrid x="10" y="160" width="476" height="152">
     <mx:columns>
       <mx:DataGridColumn headerText="Column 1" dataField="col1"/>
       <mx:DataGridColumn headerText="Column 2" dataField="col2"/>
       <mx:DataGridColumn headerText="Column 3" dataField="col3"/>
     </mx:columns>
   </mx:DataGrid>
   ```
Notice that the code defines the relative x,y position of the grid within its parent container and its width and height. The next element defines the columns with attributes for the header label and the data fields.

The goal is to achieve a table that looks like the following figure:

<table>
<thead>
<tr>
<th>Employee</th>
<th>Title</th>
<th>Monthly Salary</th>
<th>Annual Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neena Kochhar</td>
<td>Administration</td>
<td>17000</td>
<td>204000</td>
</tr>
<tr>
<td>Lex De Haan</td>
<td>Administration</td>
<td>17000</td>
<td>204000</td>
</tr>
<tr>
<td>Alexander Hunold</td>
<td>Programmer</td>
<td>9000</td>
<td>108000</td>
</tr>
<tr>
<td>Bruce Ernst</td>
<td>Programmer</td>
<td>6000</td>
<td>72000</td>
</tr>
</tbody>
</table>

2. Make the DataGrid aware of the data source by adding an attribute to the `<mx:DataGrid>` element as follows:

```
dataProvider="{dataXML.ROW}"```

This attribute defines the data object to be used at runtime to populate the grid. Remember that in this example, we defined the XML data variable as "dataXML"; now use that definition followed by "ROW" (that is, dataXML.ROW). ROW is the repeating group in the data set. Note that the syntax requires the curly braces to let the Flex engine know we are defining a data source.

3. Bind the columns. In the basic structure provided, replace the values for `dataField` with the appropriate element name from your data source. Also replace `headerText` values with the desired column heading names. For example, for the first column, replace

```
<mx:DataGridColumn headerText="Column 1" dataField="coll"/>
```

with

```
<mx:DataGridColumn headerText="Employee" dataField="NAME" />
```

This defines the first column header name as "Employee" and binds the column data to the "NAME" element in the XML data source.

The completed DataGrid sample code follows:
4. You can now preview the template with your sample data. Select Run, then Run EmployeeReport. This will open a new browser window and render the table with your sample data.

**Binding the Chart**

To bind the chart:

1. Start by bringing up the **Design** tab and highlighting the chart. Next, switch back to the **Source** view to find the chart code:

   ```xml
   <mx:ColumnChart x="10" y="10" id="columnchart1" width="476" height="142">
     <mx:series>
       <mx:ColumnSeries displayName="Series 1" yField=""/>
     </mx:series>
     <mx:Legend dataProvider="{columnchart1}" x="383" y="10"/>
   </mx:ColumnChart>
   ```

2. To bind the data source to the chart object, add the `dataProvider` attribute to the `<mx:ColumnChart>` element as follows:

   ```xml
   dataProvider="{dataXML.ROW}"  
   ```

3. Next add in the binding for the horizontal axis and the column series. This requires a little more effort. Check the Flex help files for more details. We want to create a chart showing salary by employee, similar to the following example:

   ![Salary Chart]

   To achieve this format, make the following updates to the code:

   - Add a `<horizontalAxis>` element to define the element from the data source
that will be used for the horizontal axis of the chart. Use the `categoryField` attribute to assign the data element value. In this example, the data element `NAME` is assigned.

- Modify the `<series>` group to bind the `SALARY` value to each employee `NAME` to create a bar for each employee.

Following is the sample code:

```xml
<mx:ColumnChart x="10" y="10" id="columnchart1" width="476" height="142" dataProvider="{dataXML.ROW}"
    <mx:horizontalAxis>
        <mx:CategoryAxis  categoryField="NAME" />
    </mx:horizontalAxis>
    <mx:series>
        <mx:ColumnSeries xField="NAME"  yField="SALARY" displayName="Salary"/>  
    </mx:series>
</mx:ColumnChart>
<mx:Legend dataProvider="{columnchart1}" x="383" y="10"/>
```

Note in the preceding sample, the `<mx:horizontalAxis>` element has been added and the `categoryField` attribute has the `NAME` data element assigned. This element is required to render the chart.

The `<mx:series>` element has been updated binding the `SALARY` value to each employee `NAME` to create a bar for each employee.

You do not need to update the legend code. Notice the `id` attribute of the `<mx:ColumnChart>` element matches the `dataProvider` attribute value of the `<mx:Legend>` element.

4. You can now run the template using your sample data. You should get an output showing the chart above the tabulated data as shown in the following figure:
Uploading the Flash Template to the Report Definition

To upload the template to your report definition:

1. Log in to BI Publisher and navigate to your report. Select **Edit** to launch the **Edit Report** page.

2. On the **Report** tab, select **Layouts**. Use the **Upload Template** field to locate and upload your template. To locate the correct file, use the **Browse** button and navigate to the Flex project directory. Under this directory open the bin directory and select the EmployeeReport.swf file.

3. With the **Layouts** node still selected, click **New**. Enter a name for the layout, select your template from the list, and select Flash Template as the Template Type.

4. Click **Save**. The layout will now be available for users to select to run in real time or to schedule.

Setting Properties for PDF Output

The Runtime Configuration page includes a set of properties specific to rendering Flash templates. These properties enable you to specify the size and placement of the Flash object when you select PDF as the output type.

**Important:** To produce PDF output you must specify the height and
width of the template in the Flex Builder. See Creating the Layout, page 11-6.

To set properties for the PDF output:

1. Using the Administrator or Developer role, navigate to the report that uses the Flash template and click Configure. The Runtime Configuration page includes a set of properties under the Flash heading. These properties control the placement and sizing of the Flash object in your output PDF document.

2. Enter values for the properties. Note that no properties are required. If you do not enter any values, the default values assume an 11 inch by 8.5 inch document (standard landscape), with a quarter inch inset from the upper left corner of the page as the insertion point of the Flash object. The default area in the document will be the size of the SWF object.

   • **Page width of wrapper document** – specify in points the width of the output PDF document. The default is 792, or 11 inches.

   • **Page height of wrapper document** – specify in points the height of the output PDF document. The default is 612, or 8.5 inches.

   • **Start x position of Flash area in PDF** – using the left edge of the document as the 0 axis point, specify in points the beginning horizontal position of the Flash object in the PDF document. The default is 18, or .25 inch.

   • **Start y position of Flash area in PDF** – using the upper left corner of the document as the 0 axis point, specify in points the beginning vertical position of the Flash object in the PDF document. The default is 18, or .25 inch.

   • **Width of Flash area** – enter in points the width of the area in the document for the Flash object to occupy. The default is the width of the SWF object.

   • **Height of Flash area** – enter in points the height of the area in the document for the Flash object to occupy. The default is the height of the SWF object.

**Summary**

This chapter covered the basics of adding and organizing layout objects, binding the objects to BI Publisher data sources to create Flex templates, and then loading the templates to the BI Publisher server to make them available to report users. This chapter demonstrated how to build a simple Flex template, but Adobe Flex allows you to build far more complex interactive reports for your users. The animation, "wiring" together and formatting of layout objects can be achieved with Flex. You can also summarize and create calculated fields on the incoming data. Please reference the Flex documentation for these more advanced features.
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, BI 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, BI 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. The 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:

```xml
<?xml version="1.0"?>
<!DOCTYPE OutBoundPayments:BatchRequest SYSTEM "OutBoundPayments.dtd">
<OutBoundPayments:BatchRequest>
  <PaymentsCommon:Preamble>
    <PaymentsCommon:Version>1.0.11.5.7</PaymentsCommon:Version>
  </PaymentsCommon:Preamble>
  <PaymentsCommon:TxnParties>
    <PaymentsCommon:RequesterParty>Oracle Germany</PaymentsCommon:RequesterParty>
    <PaymentsCommon:FinancialInstitution>Citi Bank</PaymentsCommon:FinancialInstitution>
  </PaymentsCommon:TxnParties>
  <PaymentsCommon:Encryption/>
  <PaymentsCommon:FileID>101803000001</PaymentsCommon:FileID>
  <PaymentsCommon:BaseRecordCount>54457</PaymentsCommon:BaseRecordCount>
</PaymentsCommon:RequestHeader>

<table>
<thead>
<tr>
<th>Level</th>
<th>Position</th>
<th>Length</th>
<th>Format</th>
<th>Pad</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>Alpha</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>Number</td>
<td>L, '0'</td>
<td>FileID</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>6</td>
<td>Alpha</td>
<td>R, '1'</td>
<td>Encryption</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>6</td>
<td>Date, YYYYMMDD</td>
<td></td>
<td>SYSDATE</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>6</td>
<td>Number</td>
<td>L, '0'</td>
<td>SEQUENCE_NUMBER</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td>Alpha</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>21</td>
<td>159</td>
<td>Alpha</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<LEVEL> Batch

<table>
<thead>
<tr>
<th>Level</th>
<th>Position</th>
<th>Length</th>
<th>Format</th>
<th>Pad</th>
<th>Data</th>
</tr>
</thead>
</table>
```
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 BI 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 12-12.

**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 12-12.
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
Following would be its table layout:

<table>
<thead>
<tr>
<th>LEVEL</th>
<th>RequestHeader</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW RECORD</td>
<td>FileHeaderRecordA</td>
</tr>
</tbody>
</table>

Data rows for the FileHeaderRecordA

<table>
<thead>
<tr>
<th>LEVEL</th>
<th>Batch</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW RECORD</td>
<td>BatchHeaderRecordA</td>
</tr>
</tbody>
</table>

Data rows for the BatchHeaderRecordA

| NEW RECORD | BatchHeaderRecordB |

Data rows for the BatchHeaderRecordB

<table>
<thead>
<tr>
<th>LEVEL</th>
<th>Payment</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW RECORD</td>
<td>PaymentRecordA</td>
</tr>
</tbody>
</table>

Data rows for the PaymentRecordA

| NEW RECORD | PaymentRecordB |
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, Payment1, Invoice1</td>
</tr>
<tr>
<td>18</td>
<td>InvoiceRecordA</td>
<td>For Batch2, Payment1, Invoice2</td>
</tr>
<tr>
<td>19</td>
<td>InvoiceRecordA</td>
<td>For Batch2, Payment1, Invoice3</td>
</tr>
<tr>
<td>Record Order</td>
<td>Record Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>20</td>
<td>BatchFooterRecordC</td>
<td>For Batch2</td>
</tr>
<tr>
<td>21</td>
<td>BatchFooterRecordD</td>
<td>For Batch2</td>
</tr>
<tr>
<td>22</td>
<td>FileFooterRecordB</td>
<td>One footer record for the EFT file</td>
</tr>
</tbody>
</table>

**New Record Command**

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

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

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

The new record command can have a name as its parameter. This name becomes the name for the record. The record name is also referred to as the record type. The name can be used in the COUNT function for counting the generated instances of the record. See COUNT, page 12-28 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 12-33 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 12-16 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>
<tr>
<td>ASC X12 Data Type</td>
<td>Format Template Data Type</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>R - Decimal number</td>
<td>Numer</td>
</tr>
<tr>
<td>TM - Time</td>
<td>Date</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:
  
  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 12-27.

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:
Mapping of Payment Format:
International Payments EFT Format

Format Setup:

<table>
<thead>
<tr>
<th>Hint: Define formatting options...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>&lt;TEMPLATE TYPE&gt;</strong></td>
</tr>
<tr>
<td>FIXEDPOSITION_BED</td>
</tr>
<tr>
<td><strong>&lt;OUTPUT CHARACTER SET&gt;</strong></td>
</tr>
<tr>
<td>ISO-8859-1</td>
</tr>
<tr>
<td><strong>&lt;NEW RECORD CHARACTER&gt;</strong></td>
</tr>
<tr>
<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:

<table>
<thead>
<tr>
<th>Hint: Define data levels that are needed in the format which do not exist in data extract...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>&lt;DEFINE LEVEL&gt;</strong></td>
</tr>
<tr>
<td>PaymentsByPayDatePayee</td>
</tr>
<tr>
<td><strong>&lt;BASE LEVEL&gt;</strong></td>
</tr>
<tr>
<td>Payment</td>
</tr>
<tr>
<td><strong>&lt;GROUPING CRITERIA&gt;</strong></td>
</tr>
<tr>
<td>‘PaymentDate, PayeeName’</td>
</tr>
<tr>
<td><strong>&lt;END DEFINE LEVEL&gt;</strong></td>
</tr>
<tr>
<td>PaymentsByPayDatePayee</td>
</tr>
</tbody>
</table>

| **<DEFINE LEVEL>**                           |
| InvoicesByReportingCatAndAttrib             |
| **<BASE LEVEL>**                             |
| Invoice                                      |
This command specifies the type of template. There are two types: FIXED_POSITION_BASED and DELIMITER_BASED.

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

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

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

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

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

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

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

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

**Define Level Command**

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

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

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

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

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

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

- **Group Sort Ascending or Group Sort Descending** - defines the sorting of the group. Insert the <GROUP SORT ASCENDING> or <GROUP SORT DESCENDING>
command row anywhere between the <DEFINE LEVEL> and <END DEFINE LEVEL> commands. The parameter of the sort command is a comma-separated list of elements by which to sort the group.

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

<table>
<thead>
<tr>
<th>Payment Instance</th>
<th>PaymentDate (grouping criterion 1)</th>
<th>PayeeName (grouping criterion 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Payment1</td>
<td>PaymentDate1</td>
<td>PayeeName1</td>
</tr>
<tr>
<td>Payment2</td>
<td>PaymentDate2</td>
<td>PayeeName1</td>
</tr>
<tr>
<td>Payment3</td>
<td>PaymentDate1</td>
<td>PayeeName2</td>
</tr>
<tr>
<td>Payment4</td>
<td>PaymentDate1</td>
<td>PayeeName1</td>
</tr>
<tr>
<td>Payment5</td>
<td>PaymentDate1</td>
<td>PayeeName3</td>
</tr>
</tbody>
</table>

In the template, construct the setup table as follows to create a level called "PaymentsByPayDatePayee" from the base level "Payment" grouped according to PaymentDate and Payee Name. Add the Group Sort Ascending command to sort each group:

```
<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>For group 1 (PaymentDate1, PayeeName1)</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>

### EFT File Structure:

- **BatchRec**
  - **PaymentGroupHeaderRec**
    - **PaymentRec**
    - **PaymentRec**
  - **PaymentGroupFooterRec**
  - **PaymentGroupFooterRec**
  - **PaymentGroupFooterRec**

### Generated output:

<table>
<thead>
<tr>
<th>Record Sequence</th>
<th>Record Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BatchRec</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PaymentGroupHeaderRec</td>
<td>For group 1 (PaymentDate1, PayeeName1)</td>
</tr>
<tr>
<td>3</td>
<td>PaymentRec</td>
<td>For Payment1</td>
</tr>
<tr>
<td>4</td>
<td>PaymentRec</td>
<td>For Payment4</td>
</tr>
<tr>
<td>5</td>
<td>PaymentGroupFooterRec</td>
<td>For group 1 (PaymentDate1, PayeeName1)</td>
</tr>
<tr>
<td>6</td>
<td>PaymentGroupHeaderRec</td>
<td>For group 2 (PaymentDate1, PayeeName2)</td>
</tr>
<tr>
<td>7</td>
<td>PaymentRec</td>
<td>For Payment3</td>
</tr>
<tr>
<td>8</td>
<td>PaymentGroupFooterRec</td>
<td>For group 2 (PaymentDate1, PayeeName2)</td>
</tr>
<tr>
<td>9</td>
<td>PaymentGroupHeaderRec</td>
<td>For group 3 (PaymentDate1, PayeeName3)</td>
</tr>
<tr>
<td>10</td>
<td>PaymentRec</td>
<td>For Payment5</td>
</tr>
<tr>
<td>11</td>
<td>PaymentGroupFooterRec</td>
<td>For group 3 (PaymentDate1, PayeeName3)</td>
</tr>
</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:

```plaintext
<DEFINE CONCATENATION> ConcatenatedInvoiceInfo

<BASE LEVEL> Invoice

<ELEMENT> InvoiceNum

<DELIMITER> ','

<END DEFINE CONCATENATION> ConcatenatedInvoiceInfo
```

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

```
car_parts_inv0001,car_parts_inv0002,car_parts_inv0003,car_parts_inv0004,car_parts_inv0005
```

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

```
TRUNCATE(ConcatenatedInvoiceInfo, 40)
SUBSTR(ConcatenatedInvoiceInfo, 1, 40)
```

Either of these statements will result in:

```
car_parts_inv0001,car_parts_inv0002,car_`
```

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

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

to get the following string:

```
parts_inv0003,car_parts_inv0004,car_par
```
Invalid Characters and Replacement Characters Commands

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

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

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

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

First name?+Last name equates to 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
Creating an eText Template

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 12-13.

**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:

IF <boolean_expressionA> THEN
    <numeric or character expression1>
[ELSIF <boolean_expressionB THEN
    <numeric or character expression2>]
...
[ELSE
    <numeric or character expression3]
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:

IF PaymentAmount/Currency/Code IN ('USD', 'EUR', 'AON', 'AZM') THEN
    PayeeAccount/FundsCaptureOrder/OrderAmount/Value * 100
ELSIF PaymentAmount/Currency/Code IN ('BHD', 'IQD', 'KWD') THEN
    PayeeAccount/FundsCaptureOrder/OrderAmount/Value * 1000
ELSE
    PayeeAccount/FundsCaptureOrder/OrderAmount/Value
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 belonging to the current (parent) level instance.

  For example, if the level to be counted is Payment and the current level is Batch, then the COUNT will return the total number of payments in the batch. However, if the current level is RequestHeader, the COUNT will return the total number of payments in the file across all batches. If the argument is a record type, the count function will count all the generated records of the (child level) record type belonging to the current level instance.

- **INTEGER_PART, DECIMAL_PART** - returns the integer or decimal portion of a numeric value. This is used in nested expressions and in commands (display condition and group by). For the final formatting of a numeric field in the data
column, use the Integer/Decimal format.

- **IS_NUMERIC** - boolean test whether the argument is numeric. Used only with the "IF" control structure.

- **TRUNCATE** - truncate the first argument - a string to the length of the second argument. If the first argument is shorter than the length specified by the second argument, the first argument is returned unchanged. This is a user-friendly version for a subset of the SQL substr() functionality.

- **SUM** - sums all the child instance of the XML extract field argument. The field must be a numeric value. The field to be summed must always be at a lower level than the level on which the SUM function was declared.

- **MIN, MAX** - find the minimum or maximum of all the child instances of the XML extract field argument. The field must be a numeric value. The field to be operated on must always be at a lower level than the level on which the function was declared.

- **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.
• **PERIODICSEQUENCE** - 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.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal to test. Used in the IF control structure only.</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Not equal to test. Used in the IF control structure only.</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than test. Used in the IF control structure only.</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than test. Used in the IF control structure only.</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to test. Used in the IF control structure only.</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to test. Used in the IF control structure only.</td>
</tr>
<tr>
<td>()</td>
<td>Function argument and expression group delimiter. The expression group inside &quot;()&quot; will always be evaluated first. &quot;()&quot; can be nested.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>Addition operator. Implicit type conversion may be performed if any of the operands are not numbers.</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction operator. Implicit type conversion may be performed if any of the operands are not numbers.</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication operator. Implicit type conversion may be performed if any of the operands are not numbers.</td>
</tr>
<tr>
<td>DIV</td>
<td>Division operand. Implicit type conversion may be performed if any of the operands are not numbers. Note that &quot;/&quot; is not used because it is part of the XPATH syntax.</td>
</tr>
<tr>
<td>Symbol</td>
<td>Usage</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>IN</td>
<td>Equal-to-any-member-of test.</td>
</tr>
<tr>
<td>NOT IN</td>
<td>Negates the IN operator.</td>
</tr>
<tr>
<td></td>
<td>Not-Equal-to-any-member-of test.</td>
</tr>
</tbody>
</table>

**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 BI Publisher engine can correctly interpret the XML elements.

There is always an extract element considered as the context element during the BI Publisher formatting process. When BI Publisher processes the data rows in a table, the level element of the table is the context element. For example, when BI 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. BI 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 BI Publisher engine to use the extract fields as the grouping criteria, not their values.
Setting Runtime Configuration Properties

Setting Runtime Properties

The Runtime Configuration page enables you to set runtime properties at the server level. You can also set properties at the report level. If conflicting values are set for a property at each level, the report level will take precedence.

To set a property at the report level, select the report, and then select the Configure link. This will launch the Runtime Configuration page, displaying a column to enable update to the properties for the report and a column that displays the read-only values set for the server.

**Note:** In versions prior to 10.1.3.2 the Runtime Configuration properties administered through this page were set in a configuration file. This file is still used as a fallback if values are not set through this interface. However, please note that the file is not updated when you update the Runtime Configuration Properties page. For details about the file, see Configuration File Reference, Oracle Business Intelligence Publisher Administrator’s and Developer’s Guide.

Bursting Properties

If you are running BI Publisher on a multiprocessor machine or even a machine with a dual core single processor, you may be able to achieve even higher bursting throughput using the multithreading functionality for bursting.

To enable multithreading for bursting, set "Enable multithreading” to true and set "Thread count” to a number greater than one up to the number of processors or cores present on the machine.

Note that if the report delivery channel is File System, there will not be any considerable performance gain using multithreading. For delivery destinations other then file delivery, you should notice the performance gain.
Due to other processes that might be running on your system you may need to empirically determine what is the optimal setting for "Thread count." Try a series of tests by varying the setting "Thread count" to see what is optimal for your environment.

Important: Leave these settings at the defaults if your system does not have multicore processors or more than one processor. Setting "Enable multithreading" to True and "Thread count" to a number greater than the number of cores on the machine will lead to higher CPU usage without any gain in performance.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Internal Name</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable multithreading</td>
<td>bursting-multithreading-on</td>
<td>false</td>
<td>Set to &quot;true&quot; to enable multithreading during bursting. This property is for use only when running BI Publisher on a machine with multiprocessors or dual core single processors.</td>
</tr>
<tr>
<td>Thread count</td>
<td>bursting-thread-count</td>
<td>2</td>
<td>If Enable multithreading is set to &quot;true&quot;, enter the number of concurrent threads you wish to have active during bursting. Do not exceed the number of processors or cores on the machine. Note that setting this value to the maximum may not necessarily achieve the best performance for your system.</td>
</tr>
</tbody>
</table>

**PDF Output Properties**

The following properties are available for PDF output:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Internal Name</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compress PDF output</td>
<td>pdf-compression</td>
<td>True</td>
<td>Specify &quot;True&quot; or &quot;False&quot; to control compression of the output PDF file.</td>
</tr>
<tr>
<td>Property Name</td>
<td>Internal Name</td>
<td>Default Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>------------------------</td>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Hide PDF viewer’s menu bars</td>
<td>pdf-hide-menu bar</td>
<td>False</td>
<td>Specify &quot;True&quot; to hide the viewer application’s menu bar when the document is active. The menu bar option is only effective when using the Export button, which displays the output in a standalone Acrobat Reader application outside of the browser.</td>
</tr>
<tr>
<td>Hide PDF viewer’s tool bars</td>
<td>pdf-hide-tool bar</td>
<td>False</td>
<td>Specify &quot;True&quot; to hide the viewer application’s toolbar when the document is active.</td>
</tr>
<tr>
<td>Replace smart quotes</td>
<td>pdf-replace-smartquotes</td>
<td>True</td>
<td>Set to &quot;False&quot; if you do not want curly quotes replaced with straight quotes in your PDF output.</td>
</tr>
</tbody>
</table>

### PDF Security

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

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Internal Name</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable PDF Security</td>
<td>pdf-security</td>
<td>False</td>
<td>If you specify &quot;True,&quot; the output PDF file will be encrypted. You must also specify the following properties:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Open document password</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Modify permissions password</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Encryption Level</td>
</tr>
<tr>
<td>Open document password</td>
<td>pdf-open-password</td>
<td>N/A</td>
<td>This password will be required for opening the document. It will enable users to open the document only. This property is enabled only when &quot;Enable PDF Security&quot; is set to &quot;True&quot;.</td>
</tr>
<tr>
<td>Modify permissions password</td>
<td>pdf-permissions-password</td>
<td>N/A</td>
<td>This password enables users to override the security setting. This property is effective only when &quot;Enable PDF Security&quot; is set to &quot;True&quot;.</td>
</tr>
<tr>
<td>Property Name</td>
<td>Internal Name</td>
<td>Default Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------------</td>
<td>---------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Encryption level</td>
<td>pdf-encryption</td>
<td>0 - low</td>
<td>Specify the encryption level for the output PDF file. The possible values are:</td>
</tr>
<tr>
<td></td>
<td>-level</td>
<td></td>
<td>• 0: Low (40-bit RC4, Acrobat 3.0 or later)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 1: High (128-bit RC4, Acrobat 5.0 or later)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This property is effective only when &quot;Enable PDF Security&quot; is set to &quot;True&quot;. When Encryption level is set to 0, you can also set the following properties:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Disable printing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Disable document modification</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Disable context copying, extraction, and accessibility</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Disable adding or changing comments and form fields</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>When Encryption level is set to 1, the following properties are available:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Enable text access for screen readers</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Enable copying of text, images, and other content</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Allowed change level</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Allowed printing level</td>
</tr>
<tr>
<td>Disable document</td>
<td>pdf-no-changing</td>
<td>False</td>
<td>Permission available when &quot;Encryption level&quot; is set to 0. When set to &quot;True&quot;, the PDF file cannot be edited.</td>
</tr>
<tr>
<td>modification</td>
<td>-the-document</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disable printing</td>
<td>pdf-no-printing</td>
<td>False</td>
<td>Permission available when &quot;Encryption level&quot; is set to 0. When set to &quot;True&quot;, printing is disabled for the PDF file.</td>
</tr>
<tr>
<td></td>
<td>-ing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Property Name</td>
<td>Internal Name</td>
<td>Default Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>---------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Disable adding or changing comments and form fields</td>
<td>pdf-no-accff</td>
<td>False</td>
<td>Permission available when &quot;Encryption level&quot; is set to 0. When set to &quot;True&quot;, the ability to add or change comments and form fields is disabled.</td>
</tr>
<tr>
<td>Disable context copying, extraction, and accessibility</td>
<td>pdf-no-cceda</td>
<td>False</td>
<td>Permission available when &quot;Encryption level&quot; is set to 0. When set to &quot;True&quot;, the context copying, extraction, and accessibility features are disabled.</td>
</tr>
<tr>
<td>Enable text access for screen readers</td>
<td>pdf-enable-acc</td>
<td>True</td>
<td>Permission available when &quot;Encryption level&quot; is set to 1. When set to &quot;True&quot;, text access for screen reader devices is enabled.</td>
</tr>
<tr>
<td>Enable copying of text, images, and other content</td>
<td>pdf-enable-cop</td>
<td>False</td>
<td>Permission available when &quot;Encryption level&quot; is set to 1. When set to &quot;True&quot;, copying of text, images, and other content is enabled.</td>
</tr>
<tr>
<td>Allowed change level</td>
<td>pdf-changes-allowed</td>
<td>0</td>
<td>Permission available when &quot;Encryption level&quot; is set to 1. Valid Values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 0: none</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 1: Allows inserting, deleting, and rotating pages</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 2: Allows filling in form fields and signing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 3: Allows commenting, filling in form fields, and signing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 4: Allows all changes except extracting pages</td>
</tr>
</tbody>
</table>
Allowed printing level | pdf-printing-allowed | 0 | Permission available when "Encryption level" is set to 1. Valid values are:
| | | | • 0: None
| | | | • 1: Low resolution (150 dpi)
| | | | • 2: High resolution

PDF Digital Signature Properties

The following properties should only be set at the report level to enable digital signature for a report and to define the placement of the signature in the output PDF document. For more information on how to enable digital signature for your output PDF documents, see Implementing a Digital Signature, *Oracle Business Intelligence Publisher Administrator’s and Developer’s Guide*.

Note that to implement digital signature for a report based on a PDF layout template or an RTF layout template, you must set the property **Enable Digital Signature** to "True" for the report.

You also must set the appropriate properties to place the digital signature in the desired location on your output report. Your choices for placement of the digital signature depend on the template type. The choices are as follows:

- **(PDF only)** Place the digital signature in a specific field by setting the **Existing signature field name** property.
- **(RTF and PDF)** Place the digital signature in a general location of the page (top left, top center, or top right) by setting the **Signature field location** property.
- **(RTF and PDF)** Place the digital signature in a specific location designated by x and y coordinates by setting the **Signature field x coordinate** and **Signature field y coordinate** properties.

  If you choose this option, you can also set **Signature field width** and **Signature field height** to define the size of the field in your document.

Note that if you enable digital signature, but do not set any location properties, the digital signature placement will default to the top left of the document.
<table>
<thead>
<tr>
<th>Property Name</th>
<th>Internal Name</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable Digital Signature</td>
<td>signature-enable</td>
<td>False</td>
<td>Set this to &quot;True&quot; to enable digital signature for the report.</td>
</tr>
<tr>
<td>Existing signature field name</td>
<td>signature-field-name</td>
<td>N/A</td>
<td>This property applies to PDF layout templates only. If your report is based on a PDF template, you can enter a field from the PDF template in which to place the digital signature. For more information on defining a field for the signature in a PDF template, see Adding or Designating a Field for a Digital Signature, page 10-18.</td>
</tr>
<tr>
<td>Signature field location</td>
<td>signature-field-location</td>
<td>top-left</td>
<td>This property can apply to RTF or PDF layout templates. This property provides a list containing the following values: Top Left, Top Center, Top Right. Choose one of these general locations and BI Publisher will insert the digital signature to the output document, sized and positioned appropriately. If you choose to set this property, do not enter X and Y coordinates or width and height properties.</td>
</tr>
<tr>
<td>Signature field X coordinate</td>
<td>signature-field-pos-x</td>
<td>0</td>
<td>This property can apply to RTF or PDF layout templates. Using the left edge of the document as the zero point of the X axis, enter the position in points that you want the digital signature to be placed from the left. For example, if you want the digital signature to be placed horizontally in the middle of an 8.5 inch by 11 inch document (that is, 612 points in width and 792 points in height), enter 306.</td>
</tr>
<tr>
<td>Property Name</td>
<td>Internal Name</td>
<td>Default Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------</td>
<td>---------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Signature field Y</td>
<td>signature-field-pos-y</td>
<td>0</td>
<td>This property can apply to RTF or PDF layout templates. Using the bottom edge of the document as the zero point of the Y axis, enter the position in points that you want the digital signature to be placed from the bottom. For example, if you want the digital signature to be placed vertically in the middle of an 8.5 inch by 11 inch document (that is, 612 points in width and 792 points in height), enter 396.</td>
</tr>
<tr>
<td>field width</td>
<td>signature-field-width</td>
<td>0</td>
<td>Enter in points (72 points equal one inch) the desired width of the inserted digital signature field. This applies only if you are also setting the properties Signature field x coordinate and Signature field Y coordinate.</td>
</tr>
<tr>
<td>Signature field height</td>
<td>signature-field-height</td>
<td>0</td>
<td>Enter in points (72 points equal one inch) the desired height of the inserted digital signature field. This applies only if you are also setting the properties Signature field x coordinate and Signature field Y coordinate.</td>
</tr>
</tbody>
</table>

### RTF Output

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

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Internal Name</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable change tracking</td>
<td>rtf-track-changes</td>
<td>False</td>
<td>Set to &quot;True&quot; to enable change tracking in the output RTF document.</td>
</tr>
<tr>
<td>Protect document for tracked changes</td>
<td>rtf-protect-document-for-tracked-changes</td>
<td>False</td>
<td>Set to &quot;True&quot; to protect the document for tracked changes.</td>
</tr>
</tbody>
</table>
### Default font

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Internal Name</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default font</td>
<td>rtf-output-default-font</td>
<td>Arial:12</td>
<td>Use this property to define the font style and size in RTF output when no other font has been defined. This is particularly useful to control the sizing of empty table cells in generated reports. Enter the font name and size in the following format <code>&lt;FontName&gt;:&lt;size&gt;</code> for example: Arial:12. Note that the font you choose must be available to the BI Publisher processing engine at runtime. See Defining Font Mappings, page 13-15 for information on installing fonts for the BI Publisher server and also for the list of fonts predefined for BI Publisher.</td>
</tr>
</tbody>
</table>

### HTML Output

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

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Internal Name</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show header</td>
<td>html-show-header</td>
<td>True</td>
<td>Set to &quot;False&quot; to suppress the template header in HTML output.</td>
</tr>
<tr>
<td>Show footer</td>
<td>html-show-footer</td>
<td>True</td>
<td>Set to &quot;False&quot; to suppress the template footer in HTML output.</td>
</tr>
<tr>
<td>Replace smart quotes</td>
<td>html-replace-smartquotes</td>
<td>True</td>
<td>Set to &quot;False&quot; if you do not want curly quotes replaced with straight quotes in your HTML output.</td>
</tr>
<tr>
<td>Character set</td>
<td>html-output-charset</td>
<td>UTF-8</td>
<td>Specify the output HTML character set.</td>
</tr>
<tr>
<td>Make HTML output accessible</td>
<td>make-accessible</td>
<td>False</td>
<td>Specify true if you want to make the HTML output accessible.</td>
</tr>
</tbody>
</table>
Use percentage width for table columns

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Internal Name</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use percentage width for table columns</td>
<td>html-output-width th-in-percentage</td>
<td>True</td>
<td>Set this property to True to render table columns according to a percentage value of the total width of the table rather than as a value in points. This property is especially useful if your browser renders tables with extremely wide columns. Setting this property to True will improve readability of the tables.</td>
</tr>
</tbody>
</table>

**FO Processing Properties**

The following properties can be set to govern FO processing:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Internal Name</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use BI Publisher’s XSLT processor</td>
<td>xslt-xsdparser</td>
<td>True</td>
<td>Controls BI Publisher’s parser usage. If set to False, XSLT will not be parsed.</td>
</tr>
<tr>
<td>Enable scalable feature of XSLT processor</td>
<td>xslt-scalable</td>
<td>False</td>
<td>Controls the scalable feature of the XDO parser. The property ”Use BI Publisher’s XSLT processor” must be set to ”True” for this property to be effective.</td>
</tr>
<tr>
<td>Enable XSLT runtime optimization</td>
<td>xslt-runtime-optimization</td>
<td>True</td>
<td>When set to ”True”, the overall performance of the FO processor is increased and the size of the temporary FO files generated in the temp directory is significantly decreased. Note that for small reports (for example 1-2 pages) the increase in performance is not as marked. To further enhance performance when you set this property to True, it is recommended that you set the property Extract attribute sets to ”False”. See RTF Template Properties, page 13-12.</td>
</tr>
<tr>
<td>Property Name</td>
<td>Internal Name</td>
<td>Default Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>---------------------</td>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Pages cached during processing</td>
<td>system-cache-page-size</td>
<td>50</td>
<td>This property is enabled only when you have specified a Temporary Directory (under General properties). During table of contents generation, the FO Processor caches the pages until the number of pages exceeds the value specified for this property. It then writes the pages to a file in the Temporary Directory.</td>
</tr>
<tr>
<td>Bidi language digit substitution type</td>
<td>digit-substitution</td>
<td>None</td>
<td>Valid values are &quot;None&quot; and &quot;National&quot;. When set to &quot;None&quot;, Eastern European numbers will be used. When set to &quot;National&quot;, Hindi format (Arabic-Indic digits) will be used. This setting is effective only when the locale is Arabic, otherwise it is ignored.</td>
</tr>
<tr>
<td>Disable variable header support</td>
<td>fo-prevent-variable-header</td>
<td>False</td>
<td>If &quot;True&quot;, prevents variable header support. Variable header support automatically extends the size of the header to accommodate the contents.</td>
</tr>
<tr>
<td>Add prefix to IDs when merging FO</td>
<td>fo-merge-conflict-resolution</td>
<td>False</td>
<td>When merging multiple XSL-FO inputs, the FO Processor automatically adds random prefixes to resolve conflicting IDs. Setting this property to &quot;True&quot; disables this feature.</td>
</tr>
<tr>
<td>Enable multithreading</td>
<td>fo-multi-threaders</td>
<td>False</td>
<td>If you have a multiprocessor machine or a machine with a dual-core single processor, you may be able to achieve faster document generation by setting this option to True. See Enabling Multithreading, page 13-12.</td>
</tr>
<tr>
<td>Disable external references</td>
<td>xdk-secure-io-mode</td>
<td>True</td>
<td>A &quot;True&quot; setting (default) disallows the importing of secondary files such as subtemplates or other XML documents during XSL processing and XML parsing. This increases the security of your system. Set this to &quot;False&quot; if your report or template calls external files. See Notes on Enabling Multithreading, page 13-12.</td>
</tr>
<tr>
<td>Property Name</td>
<td>Internal Name</td>
<td>Default Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------</td>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FO Parsing Buffer Size</td>
<td>fo-chunk-size</td>
<td>1000000</td>
<td>Sets the size of the buffer for the FO Processor. When the buffer is full, the elements from the buffer will be rendered in the report. Reports with large tables or crosstabs that require complex formatting and calculations may require a larger buffer to properly render those objects in the report. Increase the size of the buffer at the report level for these reports. Note that increasing this value will affect the memory consumption of your system.</td>
</tr>
</tbody>
</table>

**Notes on "Enable Multithreading"**

The amount of performance gain seen by enabling this setting will depend on how much the current system resources are utilized. On a system that has numerous users running and relatively high CPU utilizations, you will likely only see minor improvements after setting "Enable multithreading" to True. If the system is used by only a few users, or reports are scheduled sequentially one at a time, or the number of CPUs is more than the number of concurrent reports, then turning on multiple threads will speed up report generation.

Note that memory utilization is likely to increase once "Enable multithreading" is set to True.

**Important:** If you are running BI Publisher on a single-core, one processor machine, leave these multithreading configuration settings at the default value of False.

**RTF Template Properties**

The following properties can be set to govern RTF templates:
<table>
<thead>
<tr>
<th>Property Name</th>
<th>Internal Name</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| Extract attribute sets        | rtf-extract-attribute-sets | Auto          | The RTF processor will automatically extract attribute sets within the generated XSL-FO. The extracted sets are placed in an extra FO block, which can be referenced. This improves processing performance and reduces file size. Valid values are:  
  - Enable - extract attribute sets for all templates and subtemplates  
  - Auto - extract attribute sets for templates, but not subtemplates  
  - Disable - do not extract attribute sets |
| Enable XPath rewriting         | rtf-rewrite-path       | True          | When converting an RTF template to XSL-FO, the RTF processor will automatically rewrite the XML tag names to represent the full XPath notations. Set this property to "False" to disable this feature. |
| Characters used for checkbox  | rtf-checkbox-glyph     | Default value: Albany WT J;9746;9747/A | The BI Publisher default PDF output font does not include a glyph to represent a checkbox. If your template contains a checkbox, use this property to define a Unicode font for the representation of checkboxes in your PDF output. You must define the Unicode font number for the "checked" state and the Unicode font number for the "unchecked" state using the following syntax:  
  \texttt{\textbackslash fontname<unicode font number for true value’s glyph>;<unicode font number for false value’s glyph>}

  Example: Albany WT J;9746;9747/A |

Note that the font that you specify must be made available to BI Publisher at runtime.
### PDF Form Template Properties

The following properties can be set to govern PDF templates:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Internal Name</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remove PDF fields from output</td>
<td>remove-pdf-fields</td>
<td>False</td>
<td>Specify &quot;true&quot; to remove PDF fields from the output. When PDF fields are removed, data entered in the fields cannot be extracted. For more information, see Setting Fields as Updateable or Read Only, page 10-16.</td>
</tr>
<tr>
<td>Set all fields as read only in output</td>
<td>all-field-readonly</td>
<td>True</td>
<td>By default, BI Publisher sets all fields in the output PDF of a PDF template to be read only. If you want to set all fields to be updateable, set this property to &quot;false&quot;. For more information, see Setting Fields as Updateable or Read Only, page 10-16.</td>
</tr>
<tr>
<td>Maintain each field’s read only setting</td>
<td>all-fields-readonly-asis</td>
<td>False</td>
<td>Set this property to &quot;true&quot; if you want to maintain the &quot;Read Only&quot; setting of each field as defined in the PDF template. This property overrides the settings of &quot;Set all fields as read only in output.&quot; For more information, see Setting Fields as Updateable or Read Only, page 10-16.</td>
</tr>
</tbody>
</table>

### Flash Template Properties

The following properties can be set to govern Flash templates:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Internal Name</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page width of wrapper document</td>
<td>flash-page-width</td>
<td>792</td>
<td>Specify in points the width of the output PDF document. The default is 792, or 11 inches.</td>
</tr>
<tr>
<td>Page height of wrapper document</td>
<td>flash-page-height</td>
<td>612</td>
<td>Specify in points the height of the output PDF document. The default is 612, or 8.5 inches.</td>
</tr>
</tbody>
</table>
### Property Name | Internal Name | Default Value | Description
--- | --- | --- | ---
Start x position of Flash area in PDF | flash-startx | 18 | Using the left edge of the document as the 0 axis point, specify in points the beginning horizontal position of the Flash object in the PDF document. The default is 18, or .25 inch.

Start y position of Flash area in PDF | flash-starty | 18 | Using the upper left corner of the document as the 0 axis point, specify in points the beginning vertical position of the Flash object in the PDF document. The default is 18, or .25 inch.

Width of Flash area | flash-width | Same as flash width in points in swf | Enter in points the width of the area in the document for the Flash object to occupy. The default is the width of the SWF object.

Height of Flash area | flash-height | Same as flash height in points in swf | Enter in points the height of the area in the document for the Flash object to occupy. The default is the height of the SWF object.

---

### Defining Font Mappings

BI Publisher’s Font Mapping feature enables you to map base fonts in RTF or PDF templates to target fonts to be used in the published document. Font Mappings can be specified at the site or report level. Font mapping is performed only for PDF PowerPoint output.

There are two types of font mappings:

- **RTF Templates** - for mapping fonts from RTF templates and XSL-FO templates to PDF and PowerPoint output fonts

- **PDF Templates** - for mapping fonts from PDF templates to different PDF output fonts.
Making Fonts Available to BI Publisher

BI Publisher provides a set of Type1 fonts and a set of TrueType fonts. You can select any of the fonts in these sets as a target font with no additional setup required. For a list of the predefined fonts see BI Publisher's Predefined Fonts, page 13-17.

The predefined fonts are located in $JAVA_HOME/jre/lib/fonts. If you wish to map to another font, you must place the font in this directory to make it available to BI Publisher at runtime. If your environment is clustered, you must place the font on every server.

Setting Font Mapping at the Site Level or Report Level

A font mapping can be defined at the site level or the report level:

- To set a mapping at the site level, select the Font Mappings link from the Admin page.
- To set a mapping at the report level, select the Configuration link for the report, then select the Font Mappings tab. These settings will apply to the selected report only.

The report-level settings will take precedence over the site-level settings.

Creating a Font Mapping

From the Admin page, under Runtime Configuration, select Font Mappings.

To create a Font Mapping

- Under RTF Templates or PDF Templates, select Add Font Mapping.

- Enter the following on the Add Font Mapping page:
  - Base Font - enter the font family that will be mapped to a new font. Example: Arial
  - Select the Style: Normal or Italic (Not applicable to PDF Template font mappings)
  - Select the Weight: Normal or Bold (Not applicable to PDF Template font mappings)
  - Select the Target Font Type: Type 1 or TrueType
  - Enter the Target Font: If you selected TrueType, you can enter a specific numbered font in the collection. Enter the TrueType Collection (TTC) Number of the desired font.
BI Publisher provides a set of Type1 fonts and a set of TrueType fonts. You can select any of these fonts as a target font with no additional setup required.

The Type1 fonts are listed in the following table:

**Type 1 Fonts**

<table>
<thead>
<tr>
<th>Number</th>
<th>Font Family</th>
<th>Style</th>
<th>Weight</th>
<th>Font Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>serif</td>
<td>normal</td>
<td>normal</td>
<td>Time-Roman</td>
</tr>
<tr>
<td>1</td>
<td>serif</td>
<td>normal</td>
<td>bold</td>
<td>Times-Bold</td>
</tr>
<tr>
<td>1</td>
<td>serif</td>
<td>italic</td>
<td>normal</td>
<td>Times-Italic</td>
</tr>
<tr>
<td>1</td>
<td>serif</td>
<td>italic</td>
<td>bold</td>
<td>Times-BoldItalic</td>
</tr>
<tr>
<td>2</td>
<td>sans-serif</td>
<td>normal</td>
<td>normal</td>
<td>Helvetica</td>
</tr>
<tr>
<td>2</td>
<td>sans-serif</td>
<td>normal</td>
<td>bold</td>
<td>Helvetica-Bold</td>
</tr>
<tr>
<td>2</td>
<td>sans-serif</td>
<td>italic</td>
<td>normal</td>
<td>Helvetica-Oblique</td>
</tr>
<tr>
<td>2</td>
<td>sans-serif</td>
<td>italic</td>
<td>bold</td>
<td>Helvetica-BoldOblique</td>
</tr>
<tr>
<td>3</td>
<td>monospace</td>
<td>normal</td>
<td>normal</td>
<td>Courier</td>
</tr>
<tr>
<td>3</td>
<td>monospace</td>
<td>normal</td>
<td>bold</td>
<td>Courier-Bold</td>
</tr>
<tr>
<td>3</td>
<td>monospace</td>
<td>italic</td>
<td>normal</td>
<td>Courier-Oblique</td>
</tr>
<tr>
<td>3</td>
<td>monospace</td>
<td>italic</td>
<td>bold</td>
<td>Courier-BoldOblique</td>
</tr>
<tr>
<td>4</td>
<td>Courier</td>
<td>normal</td>
<td>normal</td>
<td>Courier</td>
</tr>
<tr>
<td>4</td>
<td>Courier</td>
<td>normal</td>
<td>bold</td>
<td>Courier-Bold</td>
</tr>
</tbody>
</table>
The TrueType fonts are listed in the following table. All TrueType fonts will be subsetted and embedded into PDF.

<table>
<thead>
<tr>
<th>Number</th>
<th>Font Family Name</th>
<th>Style</th>
<th>Weight</th>
<th>Actual Font</th>
<th>Actual Font Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Albany WT</td>
<td>normal</td>
<td>normal</td>
<td>ALBANYWT.ttf</td>
<td>TrueType (Latin1 only)</td>
</tr>
<tr>
<td>2</td>
<td>Albany WT J</td>
<td>normal</td>
<td>normal</td>
<td>ALBANWTJ.ttf</td>
<td>TrueType (Japanese flavor)</td>
</tr>
<tr>
<td>3</td>
<td>Albany WT K</td>
<td>normal</td>
<td>normal</td>
<td>ALBANWTK.ttf</td>
<td>TrueType (Korean flavor)</td>
</tr>
<tr>
<td>Number</td>
<td>Font Family Name</td>
<td>Style</td>
<td>Weight</td>
<td>Actual Font</td>
<td>Actual Font Type</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------</td>
<td>-------</td>
<td>--------</td>
<td>-------------</td>
<td>------------------</td>
</tr>
<tr>
<td>4</td>
<td>Albany VT SC</td>
<td>normal</td>
<td>normal</td>
<td>ALBANWTS.ttf</td>
<td>TrueType (Simplified Chinese flavor)</td>
</tr>
<tr>
<td>5</td>
<td>Albany VT TC</td>
<td>normal</td>
<td>normal</td>
<td>ALBANWTT.ttf</td>
<td>TrueType (Traditional Chinese flavor)</td>
</tr>
<tr>
<td>6</td>
<td>Andale Duospace WT</td>
<td>normal</td>
<td>normal</td>
<td>ADUO.ttf</td>
<td>TrueType (Latin1 only, Fixed width)</td>
</tr>
<tr>
<td>6</td>
<td>Andale Duospace WT</td>
<td>bold</td>
<td>bold</td>
<td>ADUOB.ttf</td>
<td>TrueType (Latin1 only, Fixed width)</td>
</tr>
<tr>
<td>7</td>
<td>Andale Duospace VT J</td>
<td>normal</td>
<td>normal</td>
<td>ADUOJ.ttf</td>
<td>TrueType (Japanese flavor, Fixed width)</td>
</tr>
<tr>
<td>7</td>
<td>Andale Duospace VT J</td>
<td>bold</td>
<td>bold</td>
<td>ADUOJB.ttf</td>
<td>TrueType (Japanese flavor, Fixed width)</td>
</tr>
<tr>
<td>8</td>
<td>Andale Duospace VT K</td>
<td>normal</td>
<td>normal</td>
<td>ADUOK.ttf</td>
<td>TrueType (Korean flavor, Fixed width)</td>
</tr>
<tr>
<td>8</td>
<td>Andale Duospace VT K</td>
<td>bold</td>
<td>bold</td>
<td>ADUOKB.ttf</td>
<td>TrueType (Korean flavor, Fixed width)</td>
</tr>
<tr>
<td>9</td>
<td>Andale Duospace VT SC</td>
<td>normal</td>
<td>normal</td>
<td>ADUOSC.ttf</td>
<td>TrueType (Simplified Chinese flavor, Fixed width)</td>
</tr>
<tr>
<td>9</td>
<td>Andale Duospace VT SC</td>
<td>bold</td>
<td>bold</td>
<td>ADUOSCB.ttf</td>
<td>TrueType (Simplified Chinese flavor, Fixed width)</td>
</tr>
<tr>
<td>Number</td>
<td>Font Family Name</td>
<td>Style</td>
<td>Weight</td>
<td>Actual Font</td>
<td>Actual Font Type</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------</td>
<td>-------</td>
<td>--------</td>
<td>-------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>10</td>
<td>Andale Duospace WT TC</td>
<td>normal</td>
<td>normal</td>
<td>ADUOTC.ttf</td>
<td>TrueType (Traditional Chinese flavor, Fixed width)</td>
</tr>
<tr>
<td>10</td>
<td>Andale Duospace WT TC</td>
<td>bold</td>
<td>bold</td>
<td>ADUOTCB.ttf</td>
<td>TrueType (Traditional Chinese flavor, Fixed width)</td>
</tr>
</tbody>
</table>
Supported XSL-FO Elements

The following table lists the XSL-FO elements supported in this release. 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 \texttt{inline}. Rather than list the \texttt{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-citation</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></td>
<td>marker</td>
<td></td>
</tr>
<tr>
<td></td>
<td>retrieve-marker</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>Element</td>
<td>Supported Content Elements</td>
<td>Supported Attributes</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------------</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>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>Element</td>
<td>Supported Content Elements</td>
<td>Supported Attributes</td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------------------------</td>
<td>-------------------------------</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>odd-or-even</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• odd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• even</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• any</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• inherit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>blank-or-not-blank</td>
<td></td>
<td>blank</td>
</tr>
<tr>
<td>• blank</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• not-blank</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• any</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• inherit</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>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Element</th>
<th>Supported Content Elements</th>
<th>Supported Attributes</th>
</tr>
</thead>
<tbody>
<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>list-item</td>
<td>list-item-label</td>
<td>block-properties</td>
</tr>
<tr>
<td></td>
<td>list-item-body</td>
<td></td>
</tr>
<tr>
<td>Element</td>
<td>Supported Content Elements</td>
<td>Supported Attributes</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>list-item-body</td>
<td>block, block-container, table, list-block, wrapper</td>
<td>block-properties</td>
</tr>
<tr>
<td>list-item-label</td>
<td>block, block-container, table, list-block, wrapper</td>
<td>block-properties</td>
</tr>
<tr>
<td>page-number</td>
<td>N/A</td>
<td>empty-inline-properties</td>
</tr>
<tr>
<td>page-number-citation</td>
<td>N/A</td>
<td>empty-inline-properties, ref-id</td>
</tr>
<tr>
<td><strong>Element</strong></td>
<td><strong>Supported Content Elements</strong></td>
<td><strong>Supported Attributes</strong></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></td>
<td>• auto</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• end-on-even</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• end-on-odd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• end-on-even-layout</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• end-on-odd-layout</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• no-force</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• inherit</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-referenc e</td>
<td></td>
</tr>
<tr>
<td></td>
<td>repeatable-page-master-alternative s</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>writing-mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• lr-tp</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>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>clip</td>
</tr>
<tr>
<td></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>
Converting Reports from Oracle Reports to Oracle BI Publisher

This appendix covers the following topics:

- Overview
- Prerequisites
- Running the Conversion Utility
- Uploading the PL/SQL Package to the Database
- Moving Converted Reports to the Oracle BI Publisher Reports Repository
- Testing and Editing Converted Reports

Overview

Oracle BI Publisher provides a utility for converting reports from Oracle Reports to Oracle BI Publisher.

In Oracle Reports, the data model (SQL query or extraction logic) and report layout specifications are contained in a single file. In Oracle BI Publisher the data model and the layout are separate objects. The conversion utility therefore generates several files from a single Oracle Report file that will make up your report in Oracle BI Publisher. In most cases this file set will include a PL/SQL specification and body that you will need to create in the database that contains the data for the report. The utility will also generate a report definition file and a layout template file that you will upload to the Oracle BI Publisher repository.

Once uploaded, test the report to ensure that the output is as expected and make any changes to the report as needed. Some reports will contain structures that the utility cannot convert. These must be manually implemented in the converted reports.

The overall flow for the conversion process is as follows:

1. Run the conversion utility.
2. Load the PL/SQL package into the database.

3. Upload the report to the Oracle BI Publisher repository.

4. Test the report and check the conversion log files to identify any manual modifications needed to complete the conversion.

**Prerequisites**

The following are prerequisites for running the BI Publisher Oracle Reports conversion utility:

- You must be running JDK version 1.1.8 or later.
- Your source Oracle Reports must be in XML format.
- Your CLASSPATH must include the required JAR files.

**Source Oracle Reports Must Be in XML Format**

The conversion utility requires that your source Oracle Report be in XML format. Oracle Reports XML format is supported in Oracle Reports 9i and above. If your source reports are not in Oracle Reports XML format, the conversion utility can perform this conversion automatically. For the utility to perform the conversion the following is required:

- Oracle Reports Designer 9i or later installed on the same machine as the conversion utility

  The conversion utility must be able to call the rwconverter executable to get the reports into Oracle Reports XML format.

Oracle Reports Designer is part of the Oracle Developer Suite 10g (10.1.2.0.2) available from:


**Update the Class Path with Required JAR Files**

To run the conversion utilities you need the following files defined in your class path:

- Collections.zip
- xmlparserv2-904.jar
- xdocore.jar
- aolj.jar
For customers using Oracle E-Business Suite, these libraries or the corresponding classes are available under JAVA_TOP. For customers using Oracle BI Publisher Enterprise, all these libraries are available under WEB-INF\lib. A sample path to WEB-INF\lib is as follows:

C:\Oracle\bi\oc4j\bi\j2ee\home\applications\xmlpserver\xmlpserver\WEB-INF\lib

### Running the Conversion Utility

After you have updated your class path, you can run the conversion utility from the command line. The utility is called BIPBatchConversion

Command line usage is as follows:

```java
java ... BIPBatchConversion [-debug] -source <SourceDirectory> -target <TargetDirectory> [-oraclehome <OracleHomePath>]
```

where

- **-source** - (required) is the source directory for the Oracle Reports files. All reports must be in the same format: either RDF or XML.

- **-target** - (required) is the target directory to create Oracle BI Publisher report objects. The converted objects will include the Oracle BI Publisher Report file (.xdo), the layout template file (.rtf), the PL/SQL package, and log file.

- **-oraclehome** - (conditional) If your reports are in Oracle Reports XML format do not specify this parameter.

  If your reports are in Oracle Reports RDF format, BIPBatchConversion requires rwconverter from Oracle Reports to convert the report from RDF format to XML format. Specify the Oracle home path where Oracle Report Designer (9i or later version) is installed. BIPBatchMigration assumes that rwconverter is contained in the bin directory beneath the Oracle Home path specified.

- **-debug** - (optional) Specify this parameter to run the utility in debug mode and write debug statements to the log file.

### Source Report Is an Oracle Reports RDF File

This example requires you to specify the `-oraclehome` path.

```java
java.exe -classpath D:\Jdev\project\xdocore.jar;d:\Jdev\project\collections.zip;d:\Jdev\project\aolj.jar;d:\Jdev\project\xmlparserv2-904.jar oracle.apps.xdo.rdfparser.BIPBatchConversion -source d:\reports\pay -target d:\reports\pay\output -oraclehome D:\oracle\BIToolsHome_1 -debug
```

### Source Report Is in Oracle Reports XML Format

Do not specify the `-oraclehome` parameter in this example.
java.exe -classpath
D:\Jdev\project\xdocore.jar;d:\Jdev\project\collections.zip;d:\Jdev\project\aolj.jar;d:\Jdev\project\xmlparserv2-904.jar
oracle.apps.xdo.rdfparser.BIPBatchConversion -source d:\reports\pay -target d:\reports\pay\output -debug

See the Oracle BI Publisher 10.1.3.3 Java API Reference (Javadoc) for more details on this utility and other related conversion APIs in the rdfparser package.

Output Files

The conversion utility will generate the following output files in your target directory for each converted report:

- Report definition file (format: REPORT.xdo) that includes the Data Model.

  **Note:** This file is not needed for E-Business Suite users; see the following note under Data Template.

- A data template file (format: REPORT_template.xml) that defines the data model.

  **Note:** This is not required for Oracle BI Publisher Enterprise users because the data template is embedded in REPORT.xdo.

- The default PL/SQL package specification (format: REPORTS.pls).

- Default PL/SQL package body (format: REPORTB.pls).

- The RTF Layout Template (for example: REPORT.rtf).

- A log file (format: REPORT.log).

For example, assume you have a report called invoice.rdf located in D:\reports\pay. Running this command:

java.exe -classpath
D:\Jdev\project\xdocore.jar;d:\Jdev\project\collections.zip;d:\Jdev\project\aolj.jar;d:\Jdev\project\xmlparserv2-904.jar
oracle.apps.xdo.rdfparser.BIPBatchConversion -source d:\reports\pay -target d:\reports\pay\output -oraclehome D:\oracle\BIToolsHome_1 -debug

Will generate the following output files:

- PL/SQL Package Specification: C:\BIPublisher_reports\ invoice\invoiceS.pls

- PL/SQL Package Body: C:\BIPublisher_reports\ invoice\invoiceB.pls
• Report definition file: C:\BIPublisher_reports\ invoice\invoice.xdo
• DataTemplate: C:\BIPublisher_reports\ invoice\invoice_template.xml
• RTF Layout Template: C:\BIPublisher_reports\invoice\invoice.rtf
• Log file: C:\BIPublisher_reports\invoice\invoice.log

Uploading the PL/SQL Package to the Database

Many converted Oracle Reports will generate a PL/SQL package specification file and a PL/SQL package body file as follows:

• <report_name>S.pls
• <report_name>B.pls

Run the PL/SQL package files against your Oracle Database as follows. This will create the PL/SQL package specification and body.

SQL> @C:\BIPublisher_reports\invoice\invoiceS.pls
SQL> @C:\BIPublisher_reports\invoice\invoiceB.pls

Moving Converted Reports to the Oracle BI Publisher Reports Repository

Making your reports visible in the Oracle BI Publisher repository is a two-step process:

1. Copy the report folders into the repository.
2. Refresh the repository metadata.

If you have a file-based repository, copy the report folder structure to the file system. For example, suppose your repository path was set to:

C:\oracle\bi\xmlp\XMLP

For the report in the previous example, simply copy the report directory and all its files to the desired folder structure in the existing report repository.

For example:

> copy C:\BIPublisher_reports\invoice C:\oracle\bi\xmlp\XMLP\Shared Folders\Converted reports

If you are using the XML DB functionality of the Oracle Database as your report repository, there are several ways you can upload content. One method is through WebDAV. For options and details on how to load reports into your XML DB based repository, please refer to the following topics in the Oracle XML DB Developer’s Guide:

• Loading XML Content into Oracle XML DB
• FTP, HTTP(S), and WebDAV Access to Repository Data

Once you have loaded report directories into your repository, log on to Oracle BI Publisher with administrator privileges and select "Refresh Metadata" from the System Maintenance Section of the Oracle BI Publisher Admin page. Now all reports should be available for testing and execution.

Testing and Editing Converted Reports

After you have successfully converted your reports, created the needed PL/SQL package, and moved the report definition and RTF template files into the Oracle BI Publisher repository, you should test your reports to make sure they are providing the data and formatting desired. Most converted reports will run as expected without further modification. More complex reports may require some additional modification to work as desired. Following are some common issues with converted reports.

Summary Columns Moved to the select Clause

Occasionally when converting a more complex Oracle Reports report, the Data Template or PL/SQL may contain minor errors and require manual correction. The conversion utility will move all formula columns to the select clause of the SQL query in the data model. In most cases this will not cause a problem. However, if any argument to the formula is a summary column, this will not work because the summary column will not be calculated at the time the query is executed.

To correct this problem you will need to remove this formula from the select clause and implement the formula as XSL in your layout template. Most of these formulas are used either for simple addition or summation or currency conversion, formatting, and rounding.

PL/SQL Format Trigger Logic Not Supported in RTF Layout Templates

The majority of Oracle Reports reports use simple "if" formatting logic. The conversion utility automatically converts this logic to equivalent XSL-FO and inserts the code into form fields in the RTF layout template. However, there is no support for PL/SQL in RTF layout templates. The conversion utility does not convert any PL/SQL format trigger logic present in the report. Instead the conversion utility writes all the format trigger code to a log file. You will need to implement any corresponding PL/SQL logic as XSL code.

To aid in this process, the resulting RTF template will contain form fields that hold the format trigger names that are called; these fields will be highlighted in red. You can then refer to the log to find the actual PL/SQL code used in the original Oracle Report. You will need to rewrite these PL/SQL triggers as XSL-FO.

For more information, see Extended Function Support, page 8-1.
Index

A
Adobe Flash
   - designing templates, 11-1
alignment
   - RTF template, 7-41
Analyzer for Excel, 3-11

B
background support
   - RTF templates, 7-46
barcode formatting, 7-117
bidirectional language alignment
   - RTF template, 7-41
body tags
   - PDF template, 10-8
   - RTF template, 7-15
bookmarks
   - generating PDF bookmarks from an RTF template, 7-56
   - inserting in RTF templates, 7-53
brought forward/carried forward page totals, 7-74
bursting
   - setting up, 4-30

C
calculations in PDF template, 10-13
calendar profile option, 7-114
calendar specification, 7-114
cell highlighting
   - conditional in RTF templates, 7-70
charts
   - building in RTF templates, 7-18
check box placeholder
   - creating in PDF template, 10-5
check box support
   - RTF templates, 7-57
choose statements, 7-64
clip art support, 7-29
columns
   - fixed width in tables, 7-42
conditional columns
   - rtf template, 7-65
conditional formatting, 7-61
   - table rows, 7-68
conditional formatting features, 7-61
configuration
   - setting runtime properties, 13-1
configuration properties
   - precedence of levels, 13-1
context command, 7-121
cross-tab reports, 7-94

data engine
   - api, 6-26
data models
   - options, 5-1
data template
   - calling, 6-26
   - constructing, 6-6
data template definition, 6-2
date fields in RTF templates, 7-44
digital signature
  adding signature field to a pdf template, 10-18
Discoverer
  defining data set type, 5-20
display
  setting language, 2-2
drawing support, 7-29
drop-down form field support
  RTF templates, 7-58
dynamic data columns, 7-97
  example, 7-98
dynamic table of contents in RTF template, 7-56
end on even page, 7-53
etext data tables, 12-6
etext template command rows, 12-6
etext template setup command table, 12-16
even page
  force report to end on, 7-53
Excel Analyzer, 3-11
  prerequisites, 3-11
fixed-width columns
  RTF templates, 7-42
Flash templates
  configuration properties, 11-12
  designing, 11-1
  uploading to the BI Publisher server, 11-12
FO
  supported elements, A-1
FO elements
  using in RTF templates, 7-126, 8-17
fonts
  external, 7-115
  mapping, 13-15
  setting up, 7-115
footers
  RTF template, 7-15
for-each-group XSL 2.0 standard, 7-81
formatting options in PDF templates, 10-5
form field method
  inserting placeholders, 7-8
form field properties options in PDF template, 10-5
form fields in the PDF template, 10-3
Generate XLIFF button, 9-8
groups
  basic RTF method, 7-12
  defining in PDF template, 10-7
  defining in RTF template, 7-11
  syntax, 7-11
  defining in RTF templates, 7-5
  form field method, 7-13
  grouping scenarios in RTF template, 7-11
  in RTF templates, 7-5
headers and footers
  different first page, 7-16
  different odd and even pages, 7-16
  inserting placeholders, 7-15
  multiple, 7-15
  resetting within one output file, 7-92
  RTF template, 7-15
hidden text
  support in RTF templates, 7-41
horizontal table break, 7-98
HTML output
  controlling table widths, 13-10
hyperlinks
  bookmarks, 7-53
  dynamic, 7-53
  inserting in RTF template, 7-53
  internal, 7-53
  static, 7-53
if statements, 7-62, 7-62
IF statements
  in free-form text, 7-62
if-then-else statements, 7-63
if-then-else statements, 7-63
images
  including in RTF template, 7-17
IN predicate
  If-Then-Else control structure
e-text templates, 12-28
L
language
  setting display language, 2-2
  setting report preference, 2-3
last page
  support for special content, 7-50
locale
  setting report preference, 2-3

M
markup
  adding to the PDF template, 10-3
  adding to the RTF template, 7-7
multicolumn page support, 7-45
multiple headers and footers
  RTF template, 7-15
multithreading property, 13-1

N
Namespace support in RTF template, 7-121
naming standards
  translated files, 9-6
native page breaks and page numbering, 7-40
nulls
  how to test for in XML data, 7-80

O
OLAP
  defining as a data set type, 5-22
Oracle Reports
  converting to BI Publisher
    prerequisites, B-2
    steps, B-1
output formats
  limiting by report, 4-21
overflow data
  in PDF templates, 10-17

P
page breaks
  PDF templates, 10-9
  RTF template, 7-40, 7-48
page breaks and page numbering
  native support, 7-40
page number
  setting initial
    RTF templates, 7-49
page numbers
  PDF templates, 10-9
  restarting within one output file, 7-92
  RTF template, 7-41
page totals
  brought forward/carried forward, 7-74
  inserting in RTF template, 7-72
password
  changing, 2-3
PDF template
  adding markup, 10-3
  placeholders
    types of, 10-4
pdf template mapping file, 4-29
PDF templates
  completed example, 10-14
  creating from downloaded file, 10-17
  defining groups, 10-7
  definition of, 10-1
  mapping form fields, 4-24
  overflow data, 10-17
  page breaks, 10-9
  page numbering, 10-9
  placeholders
    check box, 10-5
    radio button group, 10-6
    text, 10-4
  placement of repeating fields at runtime, 10-15
  runtime behaviors, 10-15
  sample purchase order template, 10-2
  saving as Adobe Acrobat 5.0 compatible, 10-1
  sources for document templates, 10-2
  supported modes, 10-1
  when to use, 10-1
performance
  multithreading for bursting, 13-1
placeholders
  basic RTF method, 7-7, 7-7
  form field RTF method, 7-7, 7-8
  in PDF templates, 10-3
  in RTF templates, 7-5
    defining, 7-5, 7-7
  inserting in the header and footer of RTF template, 7-15
  PDF templates
check box, 10-5
radio button group, 10-6
text, 10-4
types of, 10-4
PowerPoint output
design considerations, 7-127
predefined fonts, 13-17
preferences
setting, 2-2
setting display language, 2-2
setting passwords, 2-3
setting report locale, 2-3
SVG settings, 2-3
properties
setting at template level, 7-90

radio button group
creating in PDF templates, 10-6
regrouping, 7-81
repeating elements
See groups
report file, 9-8
report viewer height
setting, 2-3
Rich Text Format (RTF)
definition, 7-1
row breaking
preventing in RTF templates, 7-42
row formatting
conditional, 7-68
RTF placeholders
syntax, 7-7
RTF template
adding markup, 7-7
applying design elements, 7-6
definition, 7-1
designing, 7-2
groups, 7-5
including images, 7-17
native formatting features, 7-40
placeholders, 7-5
prerequisites, 7-2
sample template design, 7-3
supported modes, 7-1
basic method, 7-1
form field method, 7-1
using XSL or XSL:FO, 7-2
RTF template design
headers and footers, 7-15
RTF template placeholders, 7-7
running totals
RTF templates, 7-78

S
sample RTF template
completed markup, 7-11
section context command, 7-92
setRowsetTag method, 6-28
setRowsTag method, 6-28
setSQL method, 6-29
setting the initial page number
RTF templates, 7-49
shape support, 7-29
sorting
RTF template, 7-80
SQL functions
BI Publisher syntax for, 8-1
using in RTF templates, 7-118
SQL functions extended for BI Publisher, 8-1
text truncation, 7-43
syntax
RTF template placeholder, 7-7

T
table features
fixed-width columns, 7-42
preventing rows breaking across pages
RTF template, 7-42
text truncation, 7-43
table features
repeating table headers
RTF template, 7-42
RTF template, 7-41
table of contents support
RTF template, 7-56
dynamic TOC, 7-56
tables
controlling table widths in HTML output, 13-10

horizontal table break, 7-98
Template Builder, 4-17, 7-2
prerequisites, 4-17
text placeholder
creating in PDF template, 10-4
text truncation in tables, 7-43
timezone
set for user, 2-2
time zone
setting for reports, 2-3
totals
brought forward/carryed forward, 7-74
inserting page totals in RTF template, 7-72
running
RTF templates, 7-78
translating, 9-8
translations, 9-8
naming standards, 9-6

U
updateable variables
RTF templates, 7-87
URLs
accessing reports, 4-34

V
variables
RTF templates, 7-87
view report height
setting, 2-3

W
watermarks
RTF templates, 7-46
Web service
defining as data source, 5-11
supported formats, 5-11
Workspace
tips for using BI Publisher in, 3-20

X
XLIFF files
naming, 9-6
report file, 9-8
uploading, 9-7
XML data file
eample, 7-4
XML file
how to read, 7-4
XSL:FO elements
using in RTF templates, 7-118
XSL elements
apply a template rule, 7-124
BI Publisher syntax for , 8-15
call template, 7-125
copy the current node, 7-125
define the root element of the stylesheet, 7-125
import stylesheet, 7-125
template declaration, 7-125
using in RTF templates, 7-124
variable declaration, 7-125