# Oracle Database:Oracle Developer:Reports & BI publisher-IBM Program

**Student Guide – Volume 2**

D81924GC10

Edition 1.0

June 2013

ORACLE

# Contents (*Oracle BI Publisher 10g R3: Fundamentals*)

# 1

# Introduction to Oracle BI Publisher

# Course Objectives

After completing this course, you should be able to do the following:

- Explain the uses of Oracle BI Publisher
- Create and use a rich text format (RTF) template
- Explain the use of a portable document format (PDF) template
- Set up, configure, and use Oracle BI Publisher Enterprise
- Create reports using Extensible Markup Language (XML)
- Explore advanced features such as bursting, scheduling, and email
- Explain integration points with other Oracle products

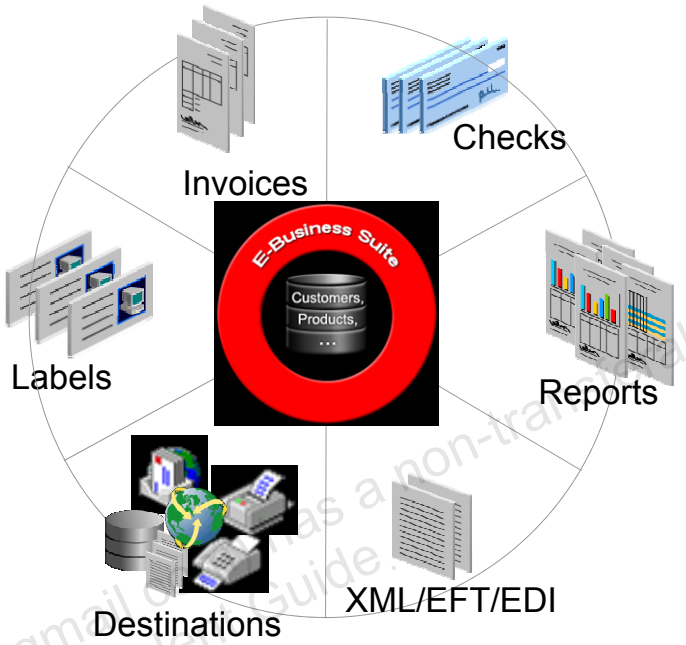
# Lesson Objectives

After completing this lesson, you should be able to do the following:

- Understand the course agenda
- Explain business document requirements and limitations of classic reporting tools
- Describe the uses of Oracle BI Publisher and its advantages over classic reporting tools
- List the key features of BI Publisher

# Oracle BI Publisher: Business Document Requirements

- Rich formatted reports
- Partner reports
- Financial statements
- Government forms
- Marketing materials
- Contracts
- Checks
- Labels
- XML/EFT/EDI
- Multiple destinations










## Oracle BI Publisher: Business Document Requirements

Oracle BI Publisher supports the spectrum of business requirements in the reporting area.

- **High fidelity reports:** Reports rich in images, charts, multiple fonts, and color are important when reports are to be distributed within and outside the company.
- **Partner reports:** Invoices, purchase orders, and checks require "rich" content and are currently highly customized.
- **Financial statements:** Your financial statements need to be presented to shareholders and upper management in a high-fidelity format.
- **Government forms:** Many government agencies now demand that you interact with them in a specific format (for example, tax forms).
- **Marketing materials:** Marketing materials generated by companies require rich personalized content.
- **Contracts:** Interacting with your customers and suppliers often requires written contracts that need to be presented in a high-fidelity format.
- **Checks:** The checks that you send to employees and suppliers require a specific format (and even ink) so that their banks are able to process them.
- **Labels:** Companies that make products need to label them (often including a barcode).
- **XML:** B2B interaction requires XML as the format. This may require a transformation from one XML format to another.
- **EFT and EDI:** Electronic funds transfer (EFT) and electronic data interchange (EDI) are formats used to communicate with your banks and partners.
- **Multiple destinations:** All these documents then need to be delivered to a report consumer (via email, fax, or printer).

# Functions of Reporting Systems

Reporting systems share three primary functions:

- Author reports
  - Business requirements definition
  - Data logic definition
  - Layout design
- Manage reports
  - Translation requirements
  - Customization needs
- Deliver reports
  - Publishing requirements (printed, email, Web page)







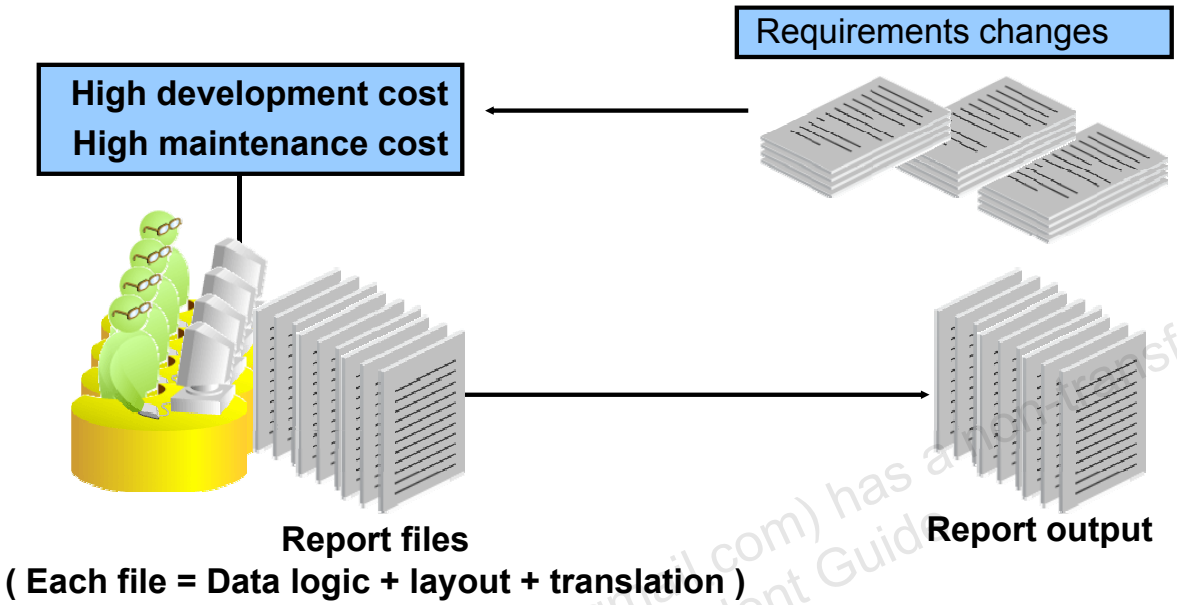
## Functions of Reporting Systems

When investing in a reporting system, a company must consider both the capabilities and costs associated with authoring, managing, and delivering required reports.

As will be illustrated in the next few slides, classic reporting tools commonly require both an initial high cost in terms of technical resources and training, and ongoing maintenance costs that may not be initially obvious.

BI Publisher meets your business needs directly out of the box. You can rapidly deploy it into your environments; it reduces your total cost of ownership and increases the return on your investment as you use the system.

# Classic Reporting Tools Paradigm






## Classic Reporting Tools Paradigm

The primary disadvantage of classic reporting tools is the high cost of ongoing maintenance.

Classic reporting approaches combine data definition (query), layout format, and translation in a single source file. Because of the nature of this file structure, the report developer must create and maintain a separate report definition file for each combination. As the data logic and report format requirements evolve, this leads to large numbers of report files. When you add the required translation, the number of report definitions becomes extremely large.

In addition to new reports requirements, companies commonly require customizations to existing reports. In this case, the only option is to take a copy of the base report definition, make the required changes, and add the modified report to the collection of report files. However, if the report development organization upgrades a base report, changes must be propagated through to the customized reports.

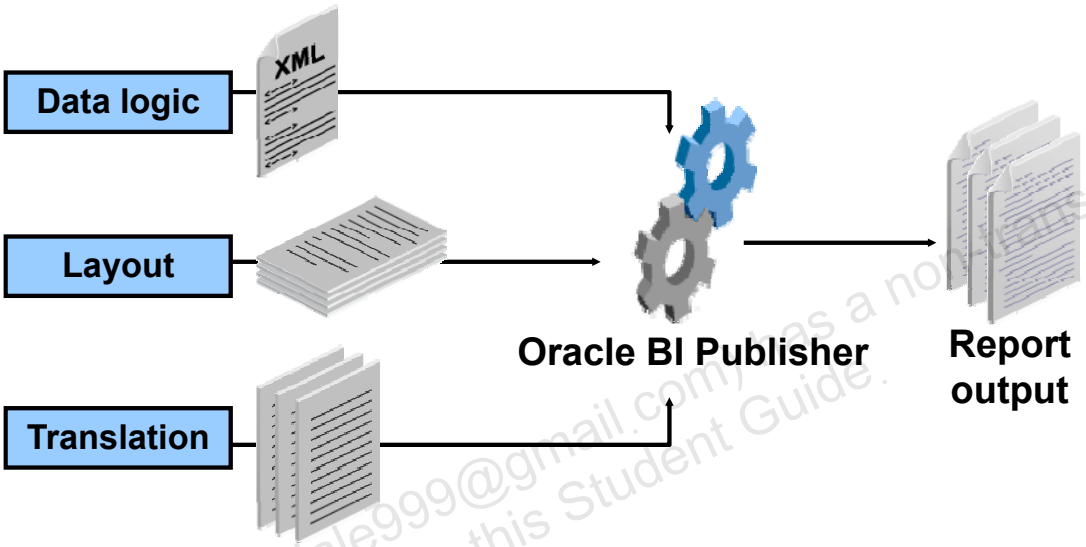
Finally, the tools used to create these reports are typically complex and require a high level of technical training to be used effectively.

These factors lead to ever-increasing complexity, and escalating maintenance and customization costs.

# Oracle BI Publisher Paradigm

Separate data, layout, and UI translation

- Flexibility
- Reduced maintenance



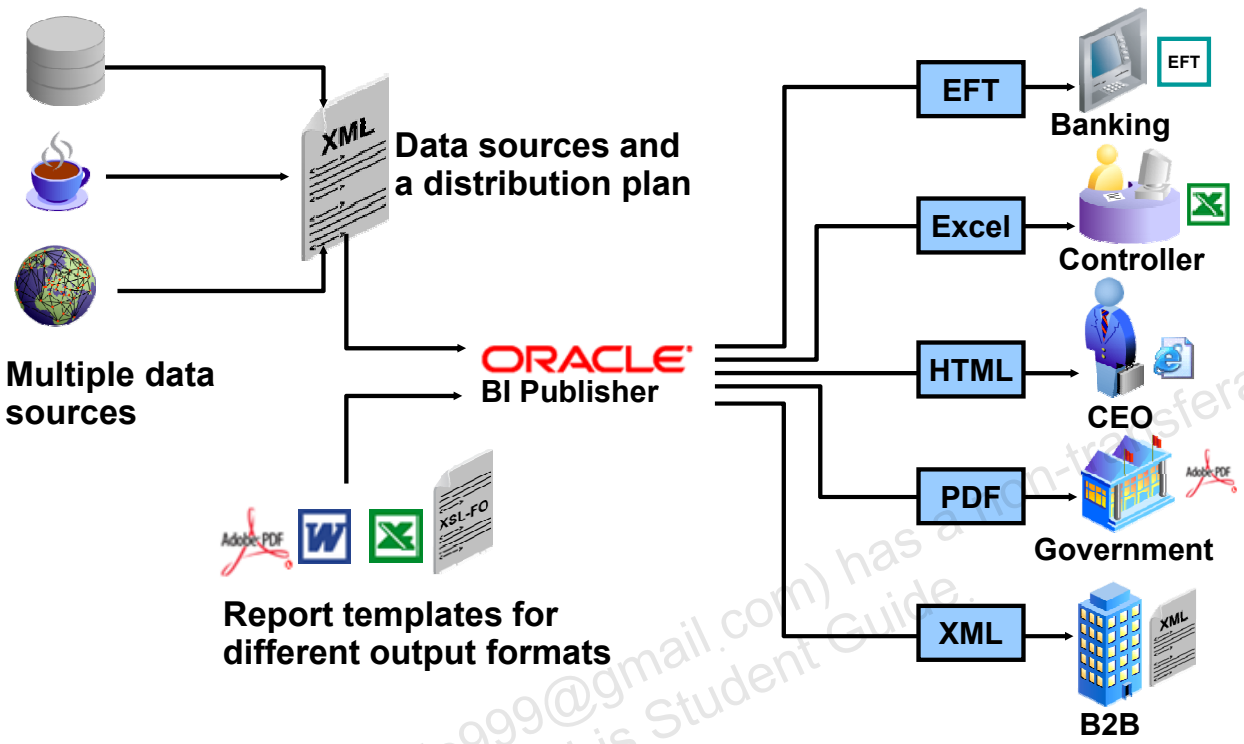



## Oracle BI Publisher Paradigm

BI Publisher breaks apart the three report-design components and treats them separately at design time. At run time, the three components are brought together by BI Publisher to generate the final formatted and translated output.

There is an immediate gain because this model is far more flexible. That is, a single data definition can support multiple layouts, and multiple translations can be applied at run time to generate translated output. This leads to a reduction in maintenance costs.

# End-to-End View of BI Publisher






## End-to-End View of BI Publisher

### Multiple Data Sources

Supported data sources include any database (or multiple databases), any XML Web service, any XML data server, RSS feeds, and external files.

### Easy Template Creation

Report templates are easily created in familiar desktop tools. With the use of different layout templates, BI Publisher simplifies the reporting architecture for customers. These templates can be used to generate multiple output formats.

### Extract Once, Publish Multiple Times

BI Publisher has the ability to publish multiple times using the same data source. For instance, a company can generate its payment data in a single data extraction program. From that program, the following can be generated:

- **Electronic funds transfer (EFT):** They are used to communicate to banks to make payments.
- **Excel:** Accountants and controllers can enter payment data into Excel for further analysis.
- **HTML:** Payment data can be published to a company portal via HTML.
- **PDF:** Government reporting requires the PDF format for communication.
- **XML:** For B2B communication, XML is quickly becoming the de facto standard.

# Salient Features of BI Publisher

- Offers support for multiple data sources
- Enables rapid deployment
  - Business consultants use familiar desktop tools for layout customization.
  - This increases developer productivity.
- Offers support for multiple output formats
- Reduces complexity
- Reduces total cost, including maintenance
- Enables flexible customization
- Follows Oracle development standards
- Enables ease of use

# Summary

In this lesson, you should have learned to:

- Describe business document requirements and limitations of classic reporting tools
- Describe the uses of Oracle BI Publisher and its advantages over classic reporting tools
- Describe the key features of BI Publisher

2

# Oracle BI Publisher: Technology and Architecture



ORACLE

# Objectives

After completing this lesson, you should be able to describe the following elements of Oracle BI Publisher:

- Components
- Architecture
- Technology

# Oracle BI Publisher

Oracle BI Publisher is a Java-based product that is available with:

- Oracle BI Server
- Oracle Applications products, including E-Business Suite, PeopleSoft, and JD Edwards

BI Publisher provides users with:

- A template-based, easy-to-use publishing solution
- A tool to rapidly develop and maintain report formats







## Oracle BI Publisher

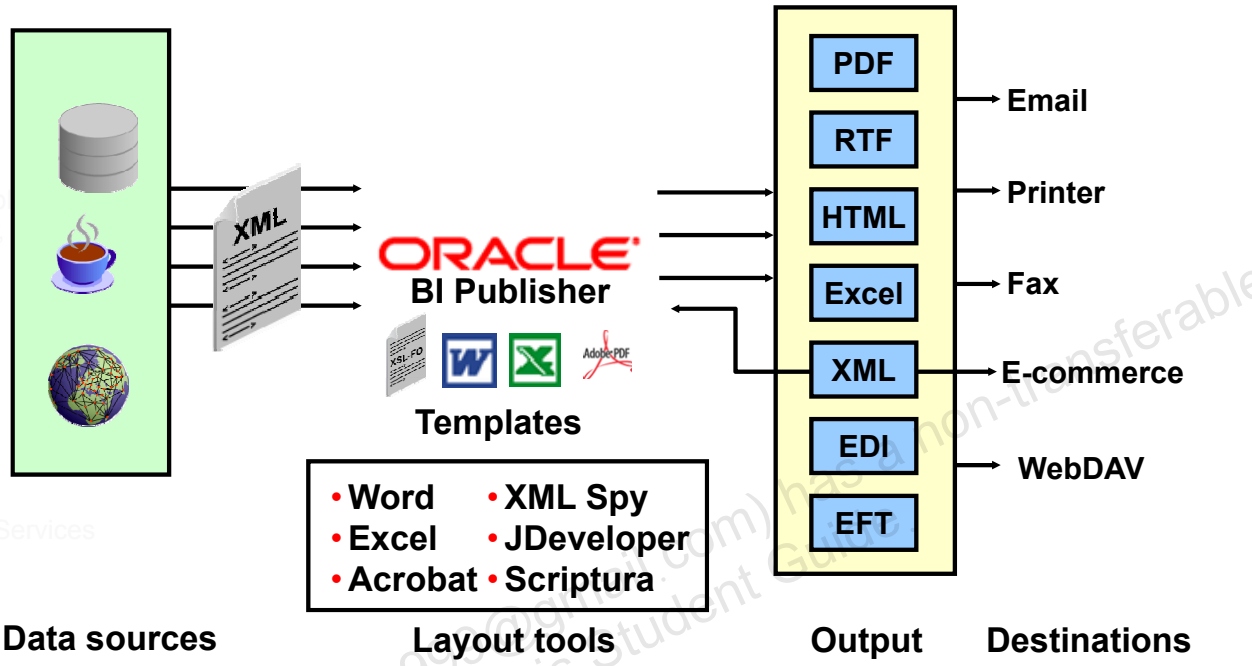
Oracle BI Publisher is a new Java-based product that is available with both Oracle BI Server and Oracle Applications products. It is an easy-to-use, template-based publishing solution that enables customers to rapidly develop and maintain report formats.

BI Publisher is built on standard, well-known technologies such as XML, and enables customers to take advantage of familiar tools such as Microsoft Word and Adobe Acrobat.

Oracle BI Publisher consists of:

- A library of APIs
- An integrated architecture with Oracle Applications products, Oracle BI EE, and Oracle BI Discoverer
- A reporting tool providing “rich” output options
- A user-friendly solution that provides control of the report design layout to the customer to decrease the overall costs associated with customization and maintenance

# Batch Processing



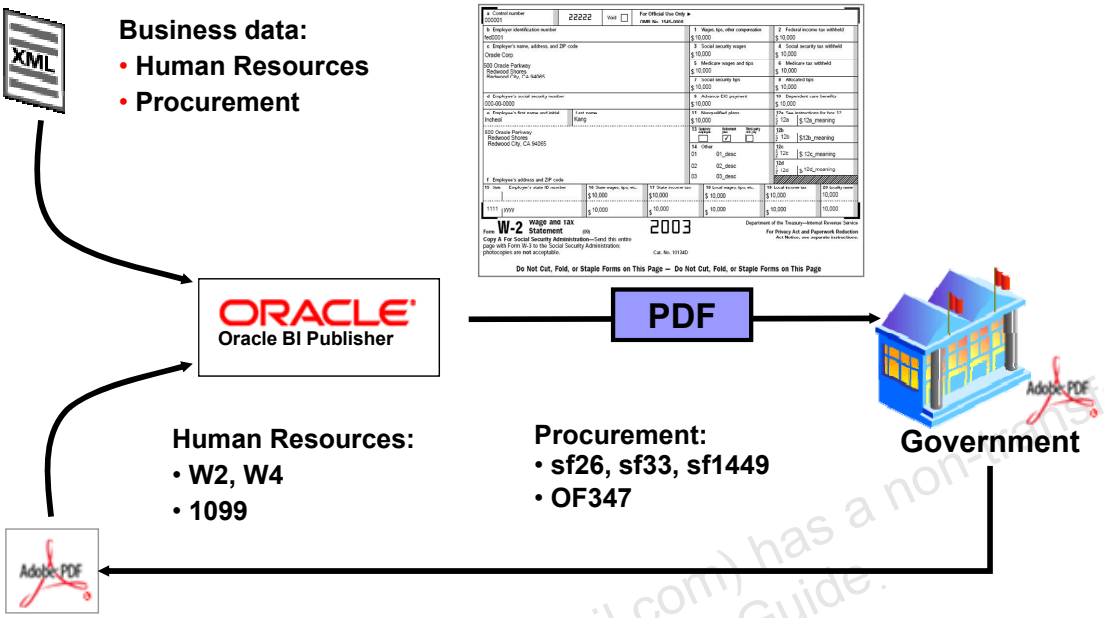



## Batch Processing

BI Publisher can process batch jobs of data from extraction to delivery.

A single file format can describe the data, the template, and the destination of individual jobs. This information is passed to the BI Publisher batch processor, and documents are generated and delivered.

# Government Forms



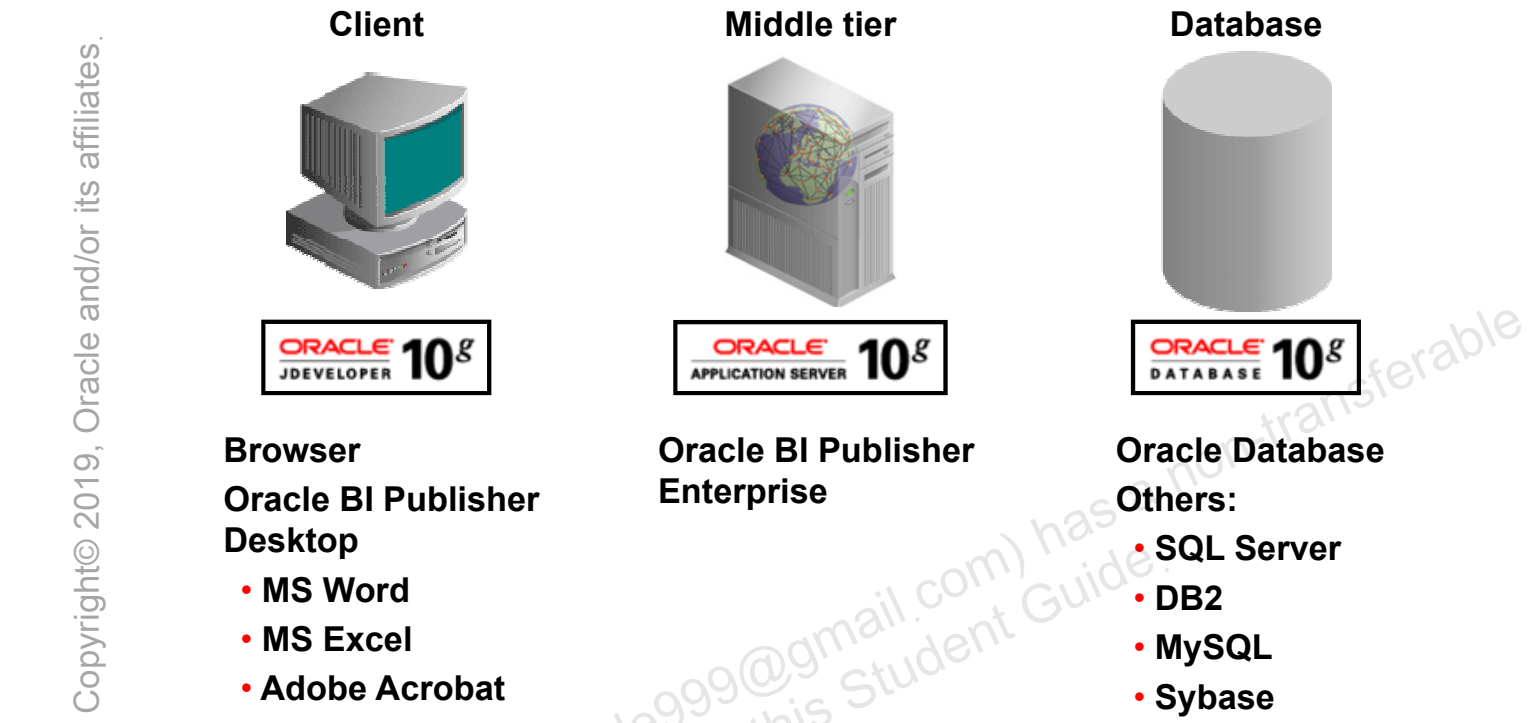

Download PDF forms from the government Web site.
Return the exact form filled with data.



## Government Forms

BI Publisher provides a closed-loop process for downloading government forms and delivering the required information to the appropriate agency.

# BI Publisher Multitier Architecture






## BI Publisher Multitier Architecture

**Client software:**

- JAVA-enabled browser
- Oracle BI Publisher desktop
- MS Word
- Ms Excel
- Adobe Acrobat

**Middle-tier software:**

Report management provided by Oracle BI Publisher Enterprise, which is integrated with:

- Oracle BI Enterprise Edition
- Oracle Applications products, including E-Business Suite, PeopleSoft, and JD Edwards

**Database software:**

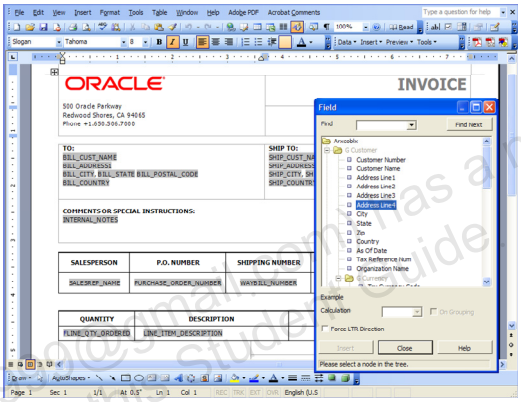
BI Publisher supports Oracle database and other types of databases.

# BI Publisher Desktop

Functional user's tools:

Users can create reports and preview results by using familiar desktop applications:

- Oracle BI Publisher for Word
- Oracle BI Publisher for Excel
- PDF editors






## BI Publisher Desktop

Oracle BI Publisher provides add-ins to two familiar desktop applications: Word and Excel.

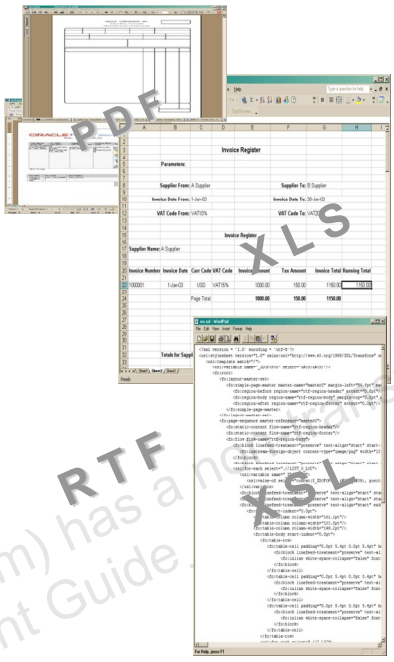
BI Publisher report formats can be designed by using tools that most users are already familiar with. Templates for BI Publisher reports are created by using Microsoft Word or Adobe Acrobat. There are no proprietary design studio components required; that is, no extra cost and no extra learning curve. BI Publisher content can also be viewed and analyzed in Excel.

### Preview Report Output

After a report template is created, you can preview the report output from within the desktop application to determine whether the report is ready for upload to BI Publisher Enterprise.

# BI Publisher Layout Templates

- Industry-standard templates
- Desktop applications:
  - Adobe Acrobat
  - MS Word
  - MS Excel
  - XSL editors
- Oracle BI Publisher Desktop Template Builder add-ins





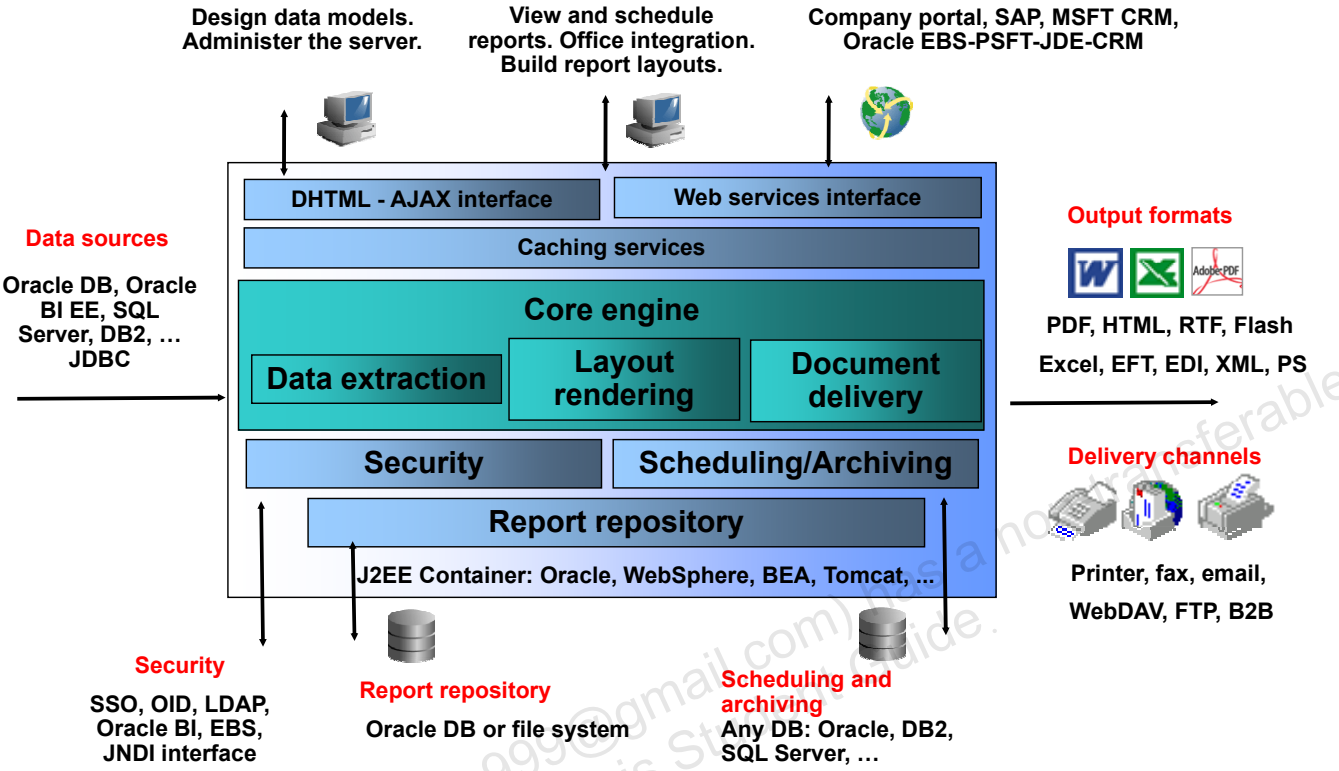



## BI Publisher Layout Templates

As mentioned previously, users can design layout templates by using familiar desktop applications such as Adobe Acrobat and Microsoft Word. A Template Builder toolbar, which is added to the desktop application interface, enables quick and easy creation of industry-standard report templates.

In addition, users can take advantage of XSL editors available on the market.

# BI Publisher Enterprise Server Architecture






## BI Publisher Enterprise Server Architecture

Oracle BI Publisher Enterprise provides the following features and benefits:

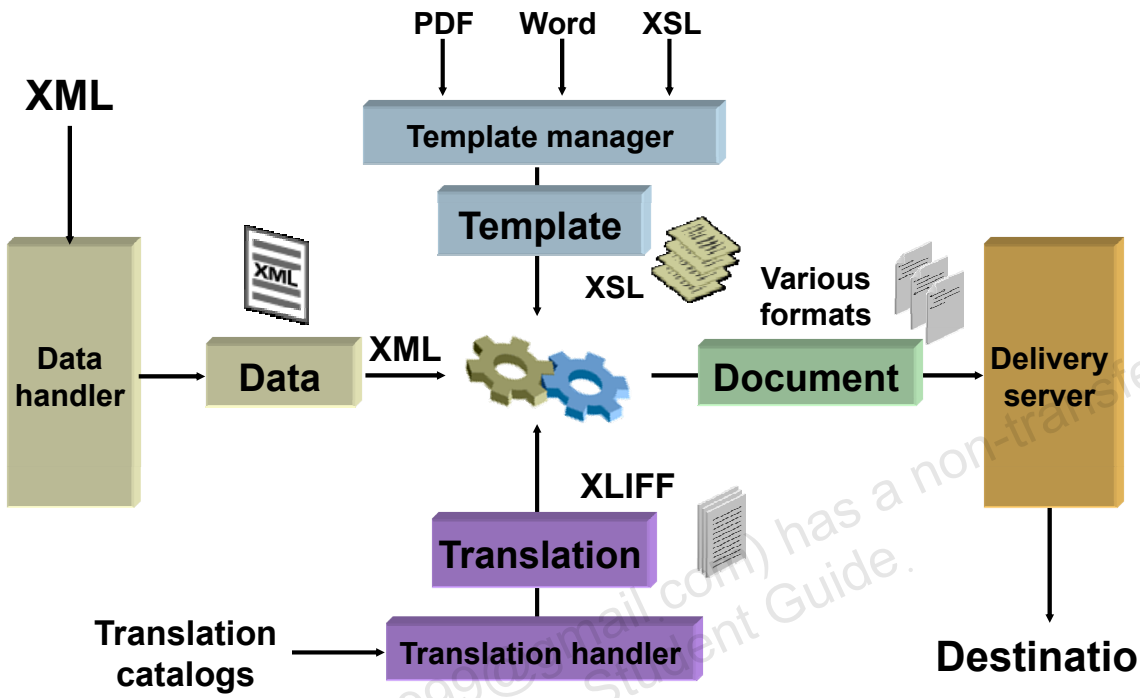
**Oracle BI Publisher Embedded: Custom development**

- Powerful formatting engines
- High-performance data extraction engine
- Batch processing and bursting
- Delivery API for email, fax, printing, FTP, and AS2
- Based on open standards: Java, XML, XSL-FO, RTF, PDF

**Oracle BI Publisher Enterprise**

- Easy and fast deployment
- Any J2EE container
- Configure repository on XML DB or file system
- Data from multiple databases in the same report (Oracle, DB2, and others)
- Multiple output formats and multiple delivery options
- Document repository, scheduling, archiving
- Database and application server independent
- Advanced security and administrative options
- Pluggable services, Web service APIs

# BI Publisher Process Architecture






## BI Publisher Process Architecture

Oracle BI Publisher is made up of five modules:

- **Data Handler:** Data engines are registered with the Data Handler. It can be any XML source or any engine that generates XML (such as Oracle Reports).
- **Template Manager:** Template Manager is a friendly user interface that is built over a collection of APIs that enables customers to register and maintain their data definitions and templates. Layout templates to be used for the final output are stored and managed in Template Manager. Templates are created by using familiar desktop tools such as Word, Excel, and Acrobat.
- **Document Processor:** With the Document Processor, a user can build a single document from multiple data source and template combinations, or create individual documents for each combination. Passing XML that contains multiple data sources and templates to Oracle BI Publisher results in the generation of multiple output documents.
- **Translation Handler:** Translation Handler provides users with the ability to register and maintain data definitions and templates in a friendly user interface.
- **Delivery Server:** Delivery Server takes the output document and delivers it to the printer.

# Data Sources

- Any database or multiple databases
  - Distributed queries
  - Oracle BI Publisher Data Engine
- Any enterprise resource planning (ERP) system
- Any XML data server
  - Oracle Reports
  - DB packages in SQL or XML
- Any XML Web service









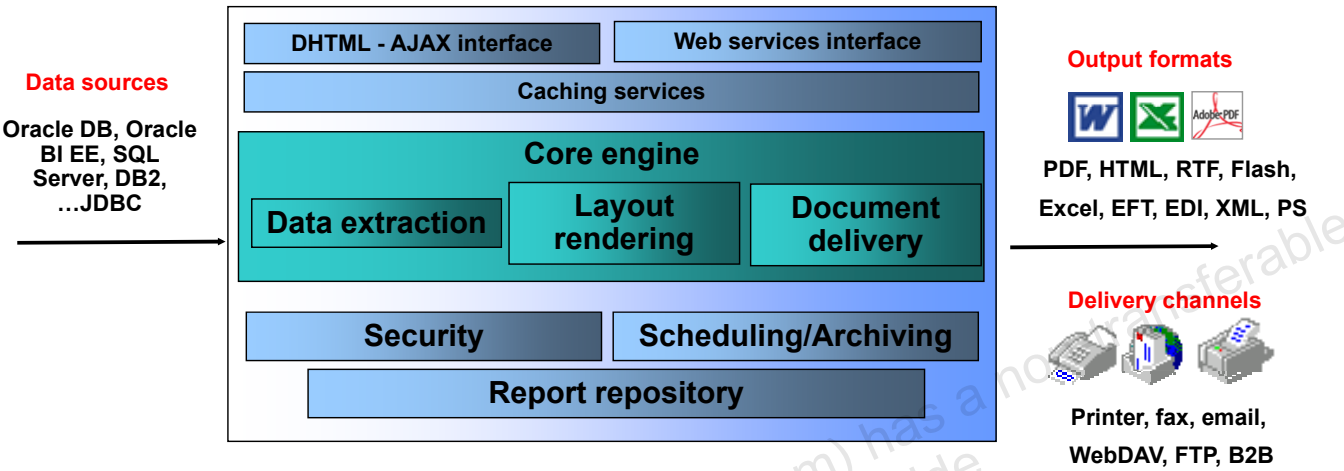


## Data Sources

Data Sources include Oracle, SQL Server, DB2, MySQL, Sybase, Web services, HTTP, JDBC, Oracle BI Enterprise Edition, and Oracle BI Discoverer.

Data can be extracted from multiple databases and applications. It may also be extracted from external systems by the BI Publisher extraction engine. These data sets can be merged either as sequential XML or merged at the line level effectively by running a query across the data sources to create a single combined hierarchical data set.

# BI Publisher Data Engine






## BI Publisher Data Engine

The BI Publisher Data Engine provides the following:

- Fast, scalable extraction engine
- Migration tools available from Oracle Reports
- Offers all that Oracle Reports offers (30% to 40% faster)
  - Multiple queries and joins
  - Event triggers
  - Java API layer for Oracle Applications Framework (OAF) support
  - Data bursting (5.6.1)
  - Distributed queries (5.6.1)

# Oracle BI Publisher Underlying Technology

Open-standard technologies:

- W3C XSL: FO implementation
- Pure Java
- Pluggable data in XML
- Output formats in PDF, RTF, and HTML
- Support for Internet Printing Protocol, WebDAV, Internet Fax Protocol, and Simple Mail Transfer Protocol (SMTP)







## Oracle BI Publisher Underlying Technology

The underlying technology is based on an implementation of the W3C XSL-FO standard. BI Publisher also provides improved performance, added security, and extensions to the FO standard to support more complex reporting requirements.

# Performance and Scalability

Stream-based implementation:

- Reduces memory footprint
- Handles large XML input files
- Is the fastest XSL-FO implementation







**Performance and Scalability**

The XSL-FO engine is not only robust but also fast. This is vital because XML data by its nature generates large objects and files, and these can cause serious memory issues during processing. BI Publisher has a stream-based implementation that reduces this memory footprint, thereby enabling large XML input files to be processed.

# Security

- Printing from MS Excel is not accepted by financial auditors.
- PDF security levels for:
  - Read-only text
  - Editable text
  - “Copyable” text
  - Printable text
  - Password-protecting text

# Internationalization and Language Support

- No need for expensive language-specific printers
- Full set of Unicode fonts supplied with BI Publisher
- Scalable fonts embedding, with CID mapping tables
- Oracle BI Publisher supports:
  - CJK
  - BiDi
  - Unicode
  - MLS


- Communication with partners around the world
- Templates created for any language or territory

ORACLE



## Language Support

In today’s global economy, the support for languages is paramount. BI Publisher ships with a full set of Unicode fonts, and a font-mapping and subsetting engine has been built. As the output is created, the engine picks the font glyphs required for the languages used. These are then embedded in the document as a new font, ensuring that the final output contains the required fonts and no specific fonts are required on the printer.

BI Publisher is unique in supporting:

- CJK
- BiDi
- Unicode
- MLS

This compares favorably to other PDF engines.

# Translation

- Communicate with partners around the world.
  - No dependency on installed languages
  - No dependency on a database character set
- Create a template for any language or territory.
  - Use XLIFF technology.
  - Provide template to translators.
- Any combination of the following are recognized:
  - 185 language codes
  - 244 territory codes







**Translation**

BI Publisher enables users to create templates in 185 languages in 244 territories. There is no dependency on the database character set. Customers can either use XLIFF technology to have their templates translated, or provide translators with the whole template. This is a great advantage because it enables translators to translate templates in "context."

# Output Formats

Oracle BI Publisher supports the following formats:

- Rich text format (RTF)
- Portable document format (PDF)
- eText (used with electronic data interchange [EDI] or electronic funds transfer [EFT])







## Output Formats

RTF is a standard maintained by Microsoft. You can learn more about it at:
http://msdn.microsoft.com/library/?url=/library/en-us/dnrtfspec/html/rtfspec.asp

PDF was created by Adobe. You can learn more about it at:
http://www.adobe.com/products/acrobat/adobepdf.html

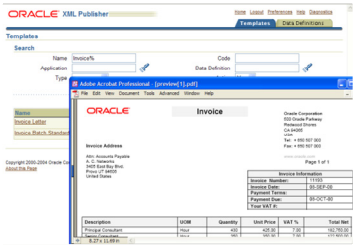
The PDF specification is available at:
http://partners.adobe.com/public/developer/pdf/index_reference.html

PDF standards in multiple output format (for example, print-based PDFs and electronic PDFs) are currently being submitted and accepted as standards to the International Standards Organization (ISO). You can find details about their published standards at:
http://www.iso.org/

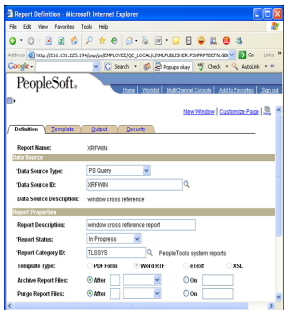
XML, XSL, XSL-FO, and XPath are standards maintained by the World Wide Web Consortium (W3C). You can find them at:
http://www.w3c.org/

# Flexible Deployment Options

## For E-Business Suite

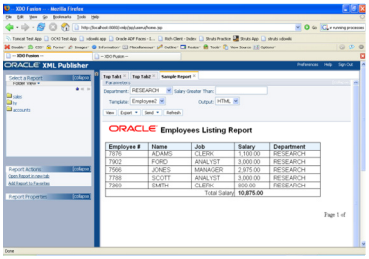


## For PeopleSoft

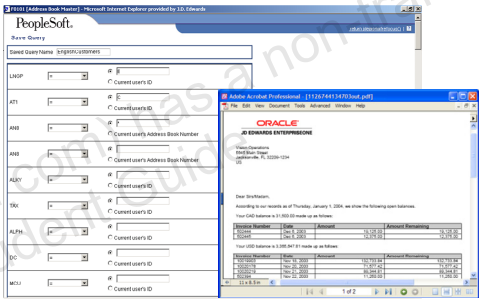


## Embedded


## For Oracle BI EE



## For JD Edwards



# Oracle BI Publisher for Applications

- Oracle BI Publisher for Oracle E-Business Suite
  - Integrated with concurrent manager
  - Tightly integrated with all E-Business Suite modules
  - Central repository for managing data and layout
  - Translation and configuration management
- Oracle BI Publisher for PeopleSoft Enterprise
  - Integrated with process scheduler and report manager
  - Publish PeopleSoft queries and row sets
  - Report bursting
  - Security join tables
  - Template manager with effective dates support
- Oracle BI Publisher for JD Edwards Enterprise
  - Integrated with Enterprise One queries
  - Support queries saved through data browser
  - Row and business unit security




ORACLE

# Summary

In this lesson, you should have learned how to describe the following elements of Oracle BI Publisher:

- Components
- Architecture
- Technology







## Summary

Oracle BI Publisher enables the following user communities to be more productive.

- **End users:** The end user is presented with rich report offerings, including multiple formats with multiple delivery options.
- **Business consultants:** Traditionally, the business consultant gathers business requirements, describes the report that is required in a document, and passes this information to the IT consultant for implementation. The business consultant now has a set of familiar desktop applications that can be used to design the report format itself, ensuring that the report meets the business requirements that are in the scope and that it looks exactly as the consultant designed it.
- **IT consultants:** The IT consultant now receives a document that is the actual report format as designed by the business consultant. The time to develop, deploy, and test is greatly reduced; the cost of ongoing maintenance is also reduced.
- **Developers:** The developer is able to focus on generating XML data to satisfy multiple requirements. Now that development is free from the "one data definition produces one report layout" paradigm, there is a move toward generating larger data engines that can satisfy a larger requirement set. This provides end users with far greater choice and control over the content of the reports they want to see.

3

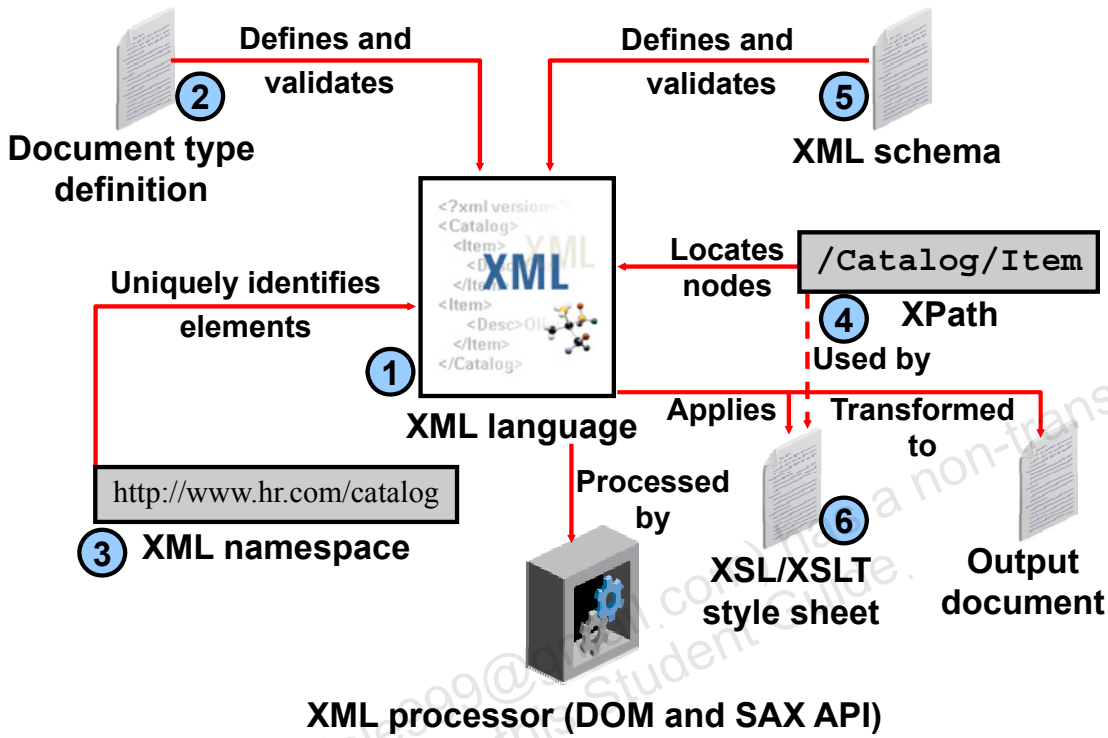
# Introduction to XML Standards

# Objectives

After completing this lesson, you should be able to do the following:

- Describe Extensible Markup Language (XML), Extensible Stylesheet Language (XSL), and other X Standards objects
- List the components of an XML document
- Create a well-formed XML document
- Describe XML namespaces
- Describe a document type definition (DTD)
- Describe XML Schema
- Describe XML Path Language (XPath)
- Describe XSL and XSL Transformations (XSLT)

# XML Standards




ORACLE







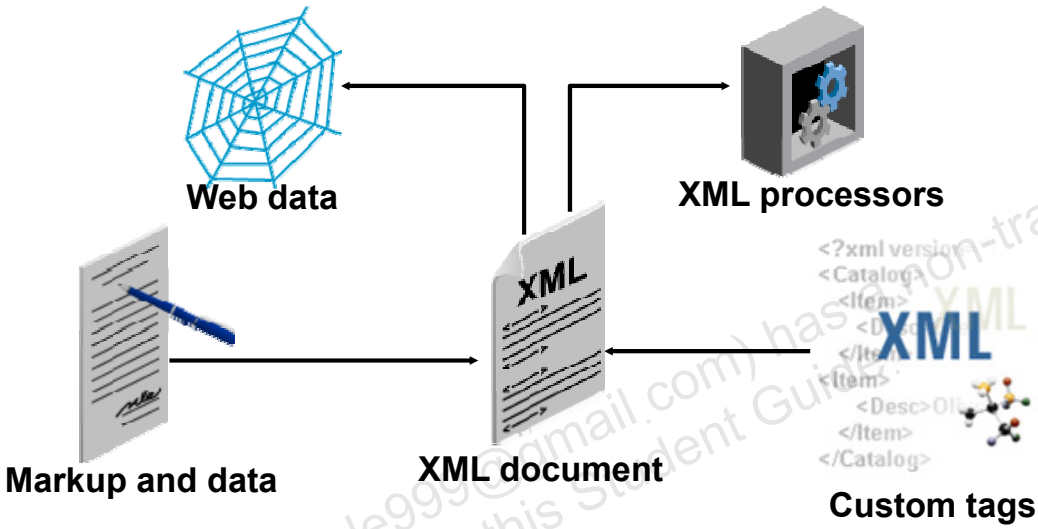
## XML Standards

The following XML standards are covered in this lesson:

1. The **XML language specification** defines the rules that govern XML document structure and how XML processors must read them.
2. A **document type definition (DTD)** provides the definition and relationships of elements contained in an XML document. A DTD validates an XML document.
3. **XML namespaces** provide a mechanism to distinguish elements with the same name but different definitions used in the same XML document.
4. The **XML Path Language (XPath)** provides the syntax for searching an XML document. XPath expressions are used in an XSL style sheet to match specific nodes.
5. An **XML Schema** provides a way to describe the XML document structure by using data type definitions, and uses namespace support. XML Schema is the preferred way to validate XML documents.
6. **Extensible Style Sheet Language (XSL)** is implemented by XSL Transformations (XSLT) to specify how to transform an XML document into another document. XSLT uses an XML vocabulary for transforming or formatting XML documents.

# Extensible Markup Language

Extensible Markup Language (XML) describes data objects called XML documents that are composed of markup and data.




ORACLE







## Extensible Markup Language

XML describes data objects called XML documents that:

- Are composed of markup language for structuring the document data
- Support custom tags for data definition, transmission, validation, and interpretation
- Have become a standard way to describe data on the Web
- Are processed by XML processors

XML was developed by an XML working group headed by the World Wide Web Consortium (W3C) with the following design goals:

- XML is usable over the Internet.
- It supports a wide variety of applications.
- It is compatible with Standard Generalized Markup Language (SGML).
- XML can be processed using easy-to-write programs.
- It has a minimum number of optional features.
- XML is human-legible and reasonably clear.
- XML enables quick design preparation.
- It enables formal and concise design.
- XML documents are easy to create.
- XML documents can be verbose.

# Advantages of Using XML

XML enables:

- A simple and extensible way to describe data
- The ability to interchange data
- Simplified business-to-business communication
- Writing of smart agents
- The ability to perform smart searches

A sample XML document:

```
<?xml version="1.0 encoding="UTF-8"?>
<books>
  <title>Building Oracle XML Applications</title>
  <title>Oracle XML Handbook</title>
  <title>Beginning XML Second Edition</title>
</books>
```







## Advantages of Using XML

XML's strongest point is its ability to perform data interchange. Because different groups of people rarely standardize on a single set of tools, it takes a significant amount of work for two groups to communicate. XML makes it easy to send structured data across the Web so that nothing is lost in translation.

When using XML, you can receive XML-tagged data from your system or another system. Neither user has to know how the other user's system is organized. If another partner or supplier teams up with your organization, you do not have to write code to exchange data with its system. You simply require them to follow the document rules defined in the DTD. You can also transform those documents by using XSLT.

When writing to an agent, one of the challenges for the agent is to make sense of incoming data. A good agent interprets information intelligently, and then responds to it accordingly. If the data sent to an agent is structured with XML, it is much easier for the agent to understand exactly what the data means and how it relates to other pieces of data that it may already know.

# Oracle XML Support

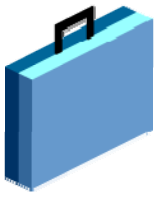
Oracle products provide support for XML in:

- Development tools


- Middle-tier frameworks


- Database storage





**Oracle XDK 10*g***

**Oracle BI Publisher**







## Oracle XML Support

**Development tools and frameworks:** Oracle XML Database (XDB), Oracle JDeveloper 10*g*, Oracle Internet File System (iFS), Oracle Application Development Framework (ADF) in JDeveloper, Oracle Portal, Oracle Application Server Reports Services, and Oracle Dynamic Services can be used to build XML applications.

**Database and middle-tier frameworks:** Oracle Application Server 10*g* provides the environment for the XSQL Pages framework, J2EE services which support XML applications, and Web services.

**Database storage:** In Oracle Database 10*g*, Oracle XDB enables XML documents to be stored in and retrieved from relational tables, `XMLType` columns, and character large object (`CLOB`) types. Oracle Text (interMedia Text) can be used to efficiently search XML documents stored in `XMLType` or `CLOB` columns.

**Oracle XML Developer's Kit (XDK) 10*g*:** Oracle XDK 10*g* is available in Oracle JDeveloper 10*g*, Oracle Application Server 10*g*, and Oracle Database 10*g*. Oracle XDK provides the following components, tools, and utilities:

- XML Parsers, XSLT Processors, XSLT VM, XML Schema processors
- XML Java Beans, XML Class Generator, XSQL Servlet, XML SQL Utility (XSU)
- XML Pipeline Processor, `TransX` Utility

# Example: A Simple XML Page

```
<?xml version="1.0"?>
<employees>
  <employee>
    <employee_id>120</employee_id>
    <last_name>Weiss</last_name>
    <salary>8000</salary>
  </employee>
  <employee>
    <employee_id>121</employee_id>
    <last_name>Fripp</last_name>
    <salary>8200</salary>
  </employee>
</employees>
```







## Example: A Simple XML Page

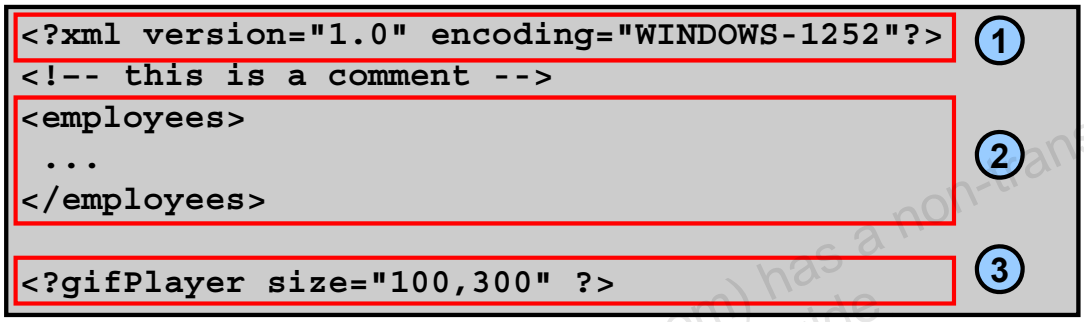
The example of a simple XML document uses nested elements to describe the employee data. Elements are identified by tag names, such as `employee`, `employee_id`, and `last_name`. Tag names are distinguishable as markup, rather than data, because they are surrounded by angle brackets (< and >).

**Note:** In XML, an element includes the start tag (`<employees>`), end tag (`</employees>`), and all markup and character data contained between the tags.

# XML Document Structure

An XML document contains the following parts:

- Prologue
- Root element
- Epilogue










## XML Document Structure

An XML document contains the following parts:

- The **prologue**, which may contain the following information:
  - XML declaration (optional in XML 1.0, mandatory in XML 1.1)
  - Document type definition (DTD), which is required only to validate the document structure
  - Processing instructions and comments, which are optional
- The **root element**, which is also called the "document element"
- An **epilogue**, which contains processing instructions and comments

An XML document can also contain processing instructions, giving commands or information to an application that is processing the XML data.

Processing instructions have the format `<?target instructions?>`, where `target` is the name of the application that is expected to do the processing, and `instructions` consist of a string of characters that embodies the information or commands for the application to process.

# XML Declaration

XML documents must start with an XML declaration.

The XML declaration:

- Looks like a processing instruction with the `xml` name, as in the following example:

```
<?xml version="1.0" encoding="WINDOWS-1252"?>
<document-root>
...
</document-root>
```

- Must contain the `version` attribute
- May include the following elements:
  - `encoding` attribute
  - `standalone` attribute
- Is optional in XML 1.0 but mandatory in XML 1.1







## XML Declaration

It is generally recommended that you start an XML document with an XML declaration. If it is present, it must be the first line in the document. Although the XML declaration looks like a processing instruction, it is really a simple declaration. In the XML 1.0 recommendation, an XML document does not require the XML declaration. However, in the XML 1.1 specifications, the XML declaration is mandatory.

**Note:** Not all tools and XML Parsers support the XML 1.1 syntax.

The only attribute that is required in the declaration is `version`. Additional attributes that can be added to the XML declaration include:

- The `encoding` attribute, which is optional. By default, XML documents are encoded in the `UTF-8` format of the Unicode character set. The slide example uses an encoding of `WINDOWS-1252`. This is useful for applications (such as those written in Java) that can handle other encoding formats.
- The `stand-alone` attribute is optional and may be set to the `yes` or `no` value. If omitted, the value is assumed to be `no`. Set the `stand-alone` value to:
  - `no` if an application requires an external DTD to determine the proper document structure
  - `yes` when the document does not have DTD or change the document content, or if the DTD is internal

# Components of an XML Document

XML documents comprise storage units containing:

- Parsed data, including the:
  - Markup (elements, attributes, entities) used to describe the data they contain
  - Character data described by markup

```
<?xml version="1.0" encoding="WINDOWS-1252"?>
<employees>
  <employee id="100">
    <name>Rachael O&apos;Leary</name>
  </employee>
</employees>
```

- Unparsed data, textual or binary information (graphic and sound data) taken as entered

```
<![CDATA[  ...unparsed data... ]]>
```







### Components of an XML Document

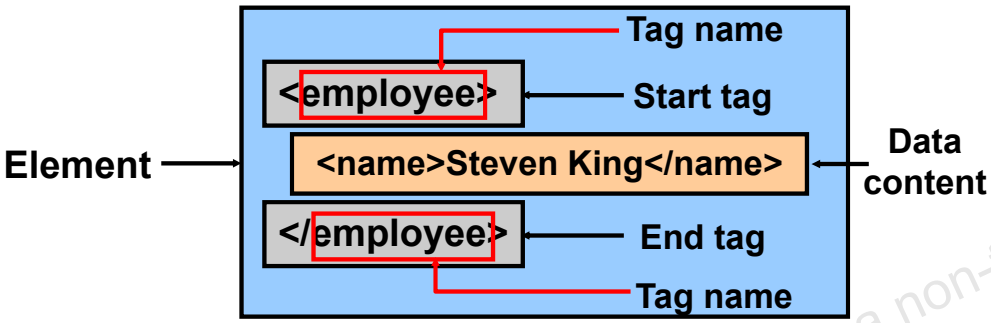
An XML document comprises storage units containing parsed or unparsed data. Parsed data is textual information comprising:

- **Markup** that describes the data it contains. Markup includes:
  - Elements to describe the data it contains, such as the root element (`employees`) and its child elements (`employee`, `name`)
  - Attributes, which are name and value pairs (`id="100"`) included in the start tag of an element
  - The entities (`&apos;`) representing any character data substituted in place of their appearance
- **Character data** described by the markup components. For example:
  - The value `100` assigned to the `id` attribute
  - The data `Rachael O'Leary` described by the `<name>` element
  - The `&apos;` entity, which represents the apostrophe (') character

**Note:** The element tree in an XML document defines its layout and logical structure. Unparsed data, embedded in `CDATA` sections, can be used in an XML document to contain textual data or encoded binary data, such as graphic and sound files. A `CDATA` section starts with **`<![CDATA[`** and ends with **`]]>`** characters. The information contained inside the `CDATA` section is not parsed by an XML Parser, and is taken as entered.

# XML Elements

- An XML element:
  - Has a start tag, end tag, and optional data content
  - Has tag names that are case-sensitive




- Empty elements:
  - Do not contain any data
  - May appear as a single tag

<initials></initials>
<initials/>







## XML Elements

An **XML element** has:

- A **start tag** (for example, `<employee>`) includes:
  - The "<"character
  - A case-sensitive tag name (`employee`), without leading spaces
  - The ">"character
- An **end tag** (for example, `</employee>`) includes:
  - The "<" character
  - The case-sensitive tag name that must be identical to the start tag name, but prefixed with a slash. Leading spaces are not permitted.
  - The ">" character
- **Data content** that can also contain elements, such as `<name>` in the slide example

In summary, an XML element includes the start tag, end tag, and everything in between.

**Empty elements** have no content between the start and end tags. In this case, a shortened form can be used where the start tag name is followed by a slash. For example: `<initials/>`.

**Tag names** are a descriptive term for an XML element and its content—for example, `employee`. The tag name is known as the *element type name*.

# Markup Rules for Elements

- There is one root element, sometimes called the top-level or document element.
- All elements:
  - Must have matching start and end tags, or be a self-closing tag (an empty element)
  - Can contain nested elements, so that their tags do not overlap
  - Have case-sensitive tag names subject to naming conventions: start with a letter, contain no spaces, and do not start with the letters `xml`
- The element data content may contain white space, such as spaces, tabs, new lines, and combinations of these.







## Markup Rules for Elements

Every XML document must contain one root element (top-level or document element). XML documents are hierarchical in structure, with elements nested in others forming a document tree. The start and end tags for elements must not overlap, as in this example:

```
<employee>
  <first_name>Steven<last_name>king</first_name></last_name>
</employee>
```

Here, the `<last_name>` element overlaps the `<first_name>` element. This is not permissible. The correct form is:

```
<employee>
  <first_name>Steven</first_name><last_name>king</last_name>
</employee>
```

Element start and end tag names must be identical—that is, they are case-sensitive. For example, `<Employee>`, `<employee>`, and `<EMPLOYEE>` are all different tag names.

Element tag names must start with a letter or an underscore (_), but not numeric or punctuation characters. After the first letter, numeric, dash (-), and period (.) characters are allowed, but not white space. Tag names cannot start with the `xml` letter sequence, or any case-sensitive combination thereof. White space, including new lines, are considered part of the data. However, XML Parsers treat end-of-line characters as a single line-feed.

# XML Attributes

An XML attribute is a name-value pair that:

- Is specified in the start tag, after the tag name

```
<?xml version="1.0" encoding="WINDOWS-1252"?>
<employees>
  <employee id="100" name='Rachael O&apos;Leary'>
    <salary>1000</salary>
  </employee>
</employees>
```

- Has a case-sensitive name
- Has a case-sensitive value that must be enclosed in matching single or double quotation marks
- Provides additional information about the XML document or XML elements





## XML Attributes

Attributes are simple name-value pairs that are associated with a particular element. XML attributes must be specified after one of the following:

- The start tag of an element
- The tag name of an empty element

Example: `<employee id="100" email="SKING"/>`

Attribute names are case-sensitive and follow the naming rules that apply to element names. In general, spaces are not used, but are allowed, on either side of the equal sign.

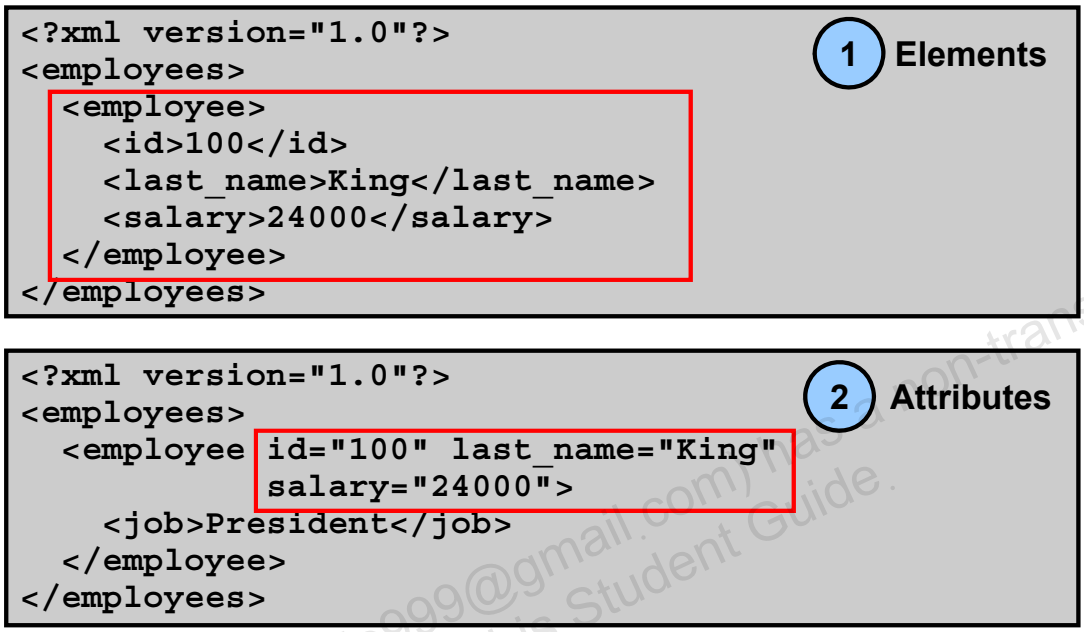
The attribute values must always be in matching quotation marks—either single or double quotation marks. The slide example shows the `employee id` attribute value enclosed in double quotation marks, and the `name` attribute value enclosed in single quotation marks. In the latter case, the `&apos;` entity must be used to include an apostrophe in the name value.

Attributes provide additional information about the XML document's content or other XML elements. Attributes can be used for the following purposes:

- Describing how the XML document data is encoded or represented
- Indicating where the links or external resources are located
- Identifying and calling external processes, such as applets and servlets
- Specifying an element instance in the document for facilitating a rapid search

**Note:** Attributes always have a value. For example, `name=""` has an empty string value.

# Using Elements Versus Attributes










## Using Elements Versus Attributes

In the slide, you see two examples. In **Example 1**, `id`, `last_name`, and `salary` are defined as elements in the `employee` element, whose parent is the `employees` element. In **Example 2**, `id`, `last_name`, and `salary` are attributes of the `employee` element, whose root element is `employees`.

Elements and attributes accomplish approximately the same thing; that is, they describe the data. The examples shown are both valid and communicate the same basic information about an employee.

**Note:** In general, it is more robust and flexible to use elements rather than attributes.

The choice to use elements or attributes is often a matter of preference, but takes some time and experience to determine the best style to use for different contexts. The easiest and the most consistent approach is to use elements for data that the user of the document wants to see, and attributes for metadata representing additional information about the data that the document may require. The following are additional points to consider:

- Elements are more easily added as your requirements expand.
- Elements may be ordered, but attributes are returned unordered from the document.
- Elements can be structured and may contain other elements.
- Attributes are atomic; that is, they cannot be nested or have attributes of their own.

# XML Entities

An XML entity:

- Is a unit of data storage
- Is identified by a case-sensitive name
- Is used as replacement text (substituted) when referencing its name between an ampersand (&) and a semicolon (;)

```
<comment>Salaries must not be &lt; 1000</comment>
```

- Has predefined names for special XML characters:
  - `<` for less-than (<) and `>` for greater-than (>)
  - `&` for ampersand (&)
  - `&quot;` for double quotation mark (")
  - `&apos;` for single quotation mark (')





## XML Entities

An XML entity is a unit of data storage that:

- Is identified by a case-sensitive name
- Represents replacement text to eliminate a lot of typing in XML documents
- Is referenced by its name prefixed with an ampersand (&) and terminated by a semicolon (;). The ampersand is referred to as an escape character.

The XML standard predefines entity names for the following special characters:

- The less-than sign (<) represented by the name `<`
- The greater-than sign (>) represented by `>`
- The ampersand (&) represented by `&`
- The double quotation mark (") represented by `&quot;`
- The single quotation mark or apostrophe (') represented by `&apos;`

In the following example, the XML Parser expects a tag name after the less-than (<) sign:

```
<number>150<100</number>
```

This invalid syntax is corrected by using the `<` built-in entity to represent the less-than character. Example:

```
<number>150&lt;100</number>
```

**Note:** User-defined entity names can be declared in a document type definition (DTD).

# XML Comments

XML comments:

- Start with `<!--`
- End with `-->`
- May appear anywhere in the character data of a document, and before the root element
- Are not elements, and can occupy multiple lines
- May not appear inside a tag or another comment

```
<?xml version="1.0" encoding="WINDOWS-1252"?>
<!-- Comment: This document has information about
     employees in the company -->
<employees>
 <name>Steven King</name> <!-- Full name -->
</employees>
```







## XML Comments

Comments are a way to document what is done inside an XML document. XML comments can appear anywhere in the XML document, except inside:

- The start or end tag of an element. Thus, they are not permissible in attribute values.
- Another comment. Comments cannot be nested.

Because comments are not considered to be XML elements, their appearance does not violate the tree structure or one-root element requirement for well-formed XML documents. The slide example shows a comment:

- Before the root element
- After the end tag of the `</name>` element

**Note:** In general, a comment must not appear before the XML declaration. However, some XML Parsers, as in Internet Explorer, do not mark it as an error.

Applications must not depend on comments because they are not intended for processing purposes, but merely for making the XML document more meaningful to people reading its contents.

# A Well-Formed XML Document

Every XML document must be well formed:

- An XML document must have one root element.
- An element must have matching start and end tag names, unless they are empty elements.
- Elements can be nested but cannot overlap.
- All attribute values must be quoted.
- Attribute names must be unique in the start tag of an element.
- Comments and processing instructions do not appear inside tags.
- The special characters `<` and `&` cannot appear in the character data of an element or attribute value.







## A Well-Formed XML Document

Every XML document must be well formed to guarantee that it is correctly structured and adheres to the rules defined in the slide. The slide contains a list of the most common rules to be addressed for well-formed documents, but it is not a complete list.

**Note:** The less-than (`<`) and ampersand (`&`) characters are special in XML and cannot appear as themselves in the character data part of an element, or the value of an attribute. In character data and attribute values, an XML Parser recognizes:

- The less-than (`<`) sign as a character that introduces the start tag of another element
- The ampersand (`&`) as an escape character before an entity name terminated by a semicolon (`;`)

Therefore, to include special characters in the character data of an element, or attribute value, you must use built-in XML entity names for special characters. For example, use `<` to include the less-than character, and `&` for the ampersand character. The XML Parser replaces the entity reference with its textual or binary representation.

**Note:** The XML 1.1 specification requires the XML declaration to appear at the beginning of the document. However, the declaration is optional when using XML 1.0.

# Class Activity

Identify errors in the following examples:

```
<?xml version="1.0"?>
<Question>Is this legal?</Question>
<Answer>No</Answer>
```
1

```
<!-- An XML document -->
<?xml version="1.0"?>
```
2

```
<Question 4You="Is this attribute name correct"/>
```
3

```
<EMAIL ID=Mark.Ant@oracle.com></EMAIL>
```
4

```
<Question>Is this legal</question>
```
5





## Class Activity

1. Both `<Question>` and `<Answer>` are top-level elements. You can have only one document element per document.
2. In general, you cannot have a comment before the XML declaration, and the root element is required.
   **Note:** Internet Explorer allows a comment before the XML declaration.
3. The attribute name cannot start with a digit.
4. The attribute value is not enclosed in quotation marks.
5. Opening and closing tag names are case-sensitive and must match.

# Comparing XML and HTML

- XML:
  - Is a markup language for describing data
  - Contains user-defined markup elements
  - Is extensible
  - Is displayed as a document tree in a Web browser
  - Conforms to rules for a well-formed document
- HTML:
  - Is a markup language for formatting data in a Web browser
  - Contains predefined markup tags
  - Is not extensible
  - Does not conform to well-formed document rules







### Comparing XML and HTML

The key difference between XML and HTML is that:

- XML is a markup language for describing data
- HTML is a markup language for formatting data

The slide covers other important differences between XML and HTML.

**A Word About XHMTL**

The World Wide Web Consortium (W3C) has worked on defining Extensible HyperText Markup Language (XHTML) as a successor to HTML.

XHTML is designed to conform to XML standards and well-formed document rules, and provide a way to reproduce, subset, and extend HTML documents. An XHTML document is a particular XML document instance intended for processing by a Web browser.

**Note:** Not all browsers support XHTML documents, and different browsers often process XHTML documents in different ways.

# XML Development

Developing XML documents can be done using:

- A simple text editor, such as Notepad
- A specialized XML editor, such as XMLSpy
- Oracle JDeveloper 10*g* XML-related features that include:
  - Syntax checking for XML documents
  - XML Editor with code insight for XML schema-driven editing
  - Registering of external XML schemas
  - Validation of XML documents against registered XML schemas







### XML Development and Oracle JDeveloper 10*g*

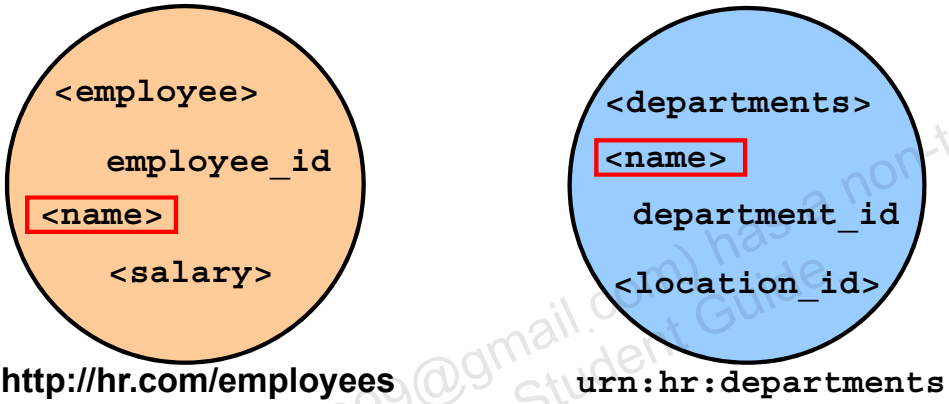
There are many ways to develop XML documents—from a simple text editor (such as Notepad) to rich XML development tools (such as XMLSpy or Oracle JDeveloper 10*g*). Oracle JDeveloper 10*g* includes:

- An XML editor, which is a specialized XML schema-driven editor for editing XML documents subject to XML language standards
- A code insight feature that enables JDeveloper to provide you with a choice of elements or attributes that are relevant to the context of the editing location in the document
- The capability to register external XML schemas for use by the schema-driven editor
- The use of registered XML schemas to validate XML documents
- A simple XML syntax check to test whether the document is well formed

# XML Namespaces

An XML namespace:

- Is identified by a case-sensitive Internationalized Resource Identifier (IRI) reference (URL or URN)
- Provides universally unique names for a collection of names (elements and attributes)










## XML Namespaces

An XML namespace:

- Is identified by an Internationalized Resource Identifier (IRI), which is a case-sensitive string of characters identifying a resource.
- Provides a universally unique name for a collection of XML names made up of element and attribute names.

The IRI is a string of characters that can be formatted as:

- A Uniform Resource Indicator (URI) or Uniform Resource Locator (URL), such as a Web address http://hr.com/employees
  **Note:** The URL, or Web address, represents a unique string and is not checked to ensure a valid or existing Web address.
- A Uniform Resource Name (URN), such as `urn:hr:departments`. A URN starts with the letters `urn`, followed by a colon (`:`), a Namespace Identifier (NID) (`hr`), a colon (`:`), and a Namespace Specific String (NSS) (`departments`).

The example in the slide shows two collections of XML element and attribute names, each identified by a unique XML namespaces IRI. If the XML namespace was not used, the `<name>` element for an employee will be indistinguishable from the `<name>` element of the department. Applying the XML namespace qualifies each `<name>` element, thereby removing the ambiguity, particularly if the documents are merged.

## XML Namespaces (continued)

### Why Use XML Namespaces?

XML namespaces are designed to provide universally unique names for elements and attributes. XML namespaces:

- Resolve name ambiguity and collision problems that can occur when XML document fragments are combined, or when multiple elements have similar type and attribute names in the same XML document
- Allow code modules to be invoked for specific elements and attributes

If two companies are interchanging XML messages, they must come to a common agreement on the meaning of the element and attribute names in the messages. This can be achieved by two means:

- Defining the meaning, format, and domain of every element and attribute needed
- Recognizing element and attribute names without ambiguity

The first can be accomplished by using an XML Schema definition. The second is solved by using XML namespaces to qualify element names.

The examples in the slide illustrate the combining of a `<department>` element with one of its `<employee>` elements. Each document is different and defines a `<name>` element. Each `<name>` element has a different content model and must be interpreted by an application in a different way. The problem does not arise when the elements exist in separate documents.

# XML Namespace Declarations: Example

```
<?xml version="1.0"?>
<department xmlns="urn:hr:department-ns"
            xmlns:emp="urn:hr:employee-ns">
  <name>Executive</name>
  <emp:employee>
    <emp:name>
      <emp:first_name>Steven</emp:first_name>
      <emp:last_name>King</emp:last_name>
    </emp:name>
  </emp:employee>
  <emp:employee>
    <emp:name>
      <emp:first_name>Neena</emp:first_name>
      <emp:last_name>Kochhar</emp:last_name>
    </emp:name>
  </emp:employee>
</department>
```

ORACLE







## XML Namespace Declarations: Example

The slide shows an example of a `<department>` element containing two `<employee>` elements. The `<department>` element shows an example of declaring two XML namespaces: a default and one with the `emp` prefix. The `<name>` element is used for:

- The department name containing the `Executives` text
- The employee name containing the `<first_name>` and `<last_name>` child elements

Without using XML namespaces in the document, the `<name>` element will be ambiguous to a processor and can possibly be treated as the same type, even though the department and employee names are semantically and structurally different.

Using the XML namespaces removes the ambiguity for each `<name>` element, which allows them to be processed differently by an XML application. In the example in the slide:

- The unqualified `<department>` and `<name>` elements are implicitly qualified by the default namespace, `urn:hr:department-ns`, declared in the start tag of the `<department>` element
- All `<employee>` elements and their children, including the `<emp:name>` element, are explicitly qualified with the `emp` prefix, which is associated with the `urn:hr:employee-ns` XML namespace

# Why Validate an XML Document?

- Well-formed documents satisfy XML syntax rules, and not the business requirements about the content and structure.
- Business rules often require validation of the content and structure of a document.
- XML documents must satisfy structural requirements imposed by the business model.
- A valid XML document can be reliably processed by XML applications.
- Validations can be done by using a DTD or an XML Schema.





ORACLE



## Why Validate an XML Document?

A well-formed XML document is one that meets all the rules of the XML specification. What if just following the syntax rules is not quite good enough? Your business may require the validation of the actual structure and content of a document. Your document must not only follow the XML rules, but it must also satisfy structural requirements imposed by your business model, as in the following example:

> Each `<employee>` element must consist of `<employee_id>`, `<first_name>`, `<last_name>`, `<email>`, `<phone_number>`, and `<department_id>` elements. If an `<employee>` element misses any of these elements, it is considered invalid. You may also need to verify whether these elements have a valid value.

You can perform these validations by using a document type definition (DTD). DTD was the first type of mechanism available for validating XML document structure and content. It lacked the capability of performing data type validations on the content. The XML Schema recommendation supports validation for different data types, and is rapidly replacing DTD as a way to validate the XML document structure and content.

**Note:** Theoretically, anything that has not been explicitly permitted in DTD is forbidden. Some parsers do not always enforce the rules defined by a DTD.

# Document Type Definition

A document type definition (DTD):

- Is the grammar for an XML document
- Contains the definitions of:
  - Elements
  - Attributes
  - Entities
  - Notations
- Contains specific instructions that the XML parser interprets to check document validity
- May be stored in a separate file (external)
- May be included in the document (internal)







## Document Type Definition

A DTD provides a list of elements contained in an XML document that collectively specifies the structure of the document. A DTD defines the set of markup items used in an XML document—that is, the:

- Elements
- Attributes, and their permissible values
- Entities (user-defined)
- Notations

**Note:** Notations are not frequently used.

The XML engine interprets the markup defined in the DTD to check the document for validity. For example, a DTD may specify that a department must have exactly one department identification number and one name, but may have multiple employee elements.

An XML document indicates to an XML Parser whether it is associated with a DTD and where to locate the DTD.

A DTD may be found internally (inline) in the document, or externally identified by a Uniform Resource Locator (URL), which can be a file on disk. If the XML Parser does not encounter errors, the XML document is guaranteed to be consistent with the definition.

# Simple DTD Declaration: Example

Example of a simple DTD with element declarations:

```
<!ELEMENT employees (employee)>
<!ELEMENT employee (name)>
<!ELEMENT name (#PCDATA)>
```

A valid XML document based on the DTD is:

```
<?xml version="1.0"?>
<employees>
  <employee>
    <name>Steven King</name>
  </employee>
</employees>
```

Note: All child elements must be defined.







### Simple DTD Declaration: Example

In the slide, the DTD declares three elements: `employees`, `employee`, and `name`. These elements form a simple hierarchy, where the `<employees>` element must contain one `<employee>` element that must contain one `<name>` element (as shown in the slide).

**Note**

- The `#PCDATA` example represents text-only content without child elements.
- The XML document does not reference the DTD in the slide example.

Element names used in the content model represent a single instance of that element. Using the DTD in the slide, the following XML document is invalid because there are two `<employee>` child elements:

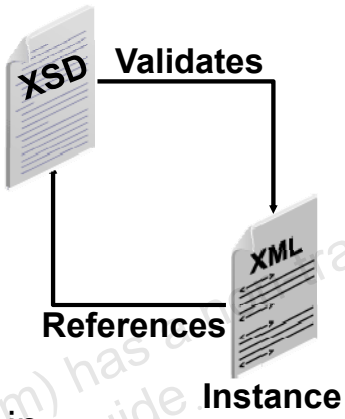
```
<?xml version="1.0"?>
<employees>
  <employee><name>Steven King</name></employee>
  <employee><name>Ellen Abel</name></employee>
</employees>
```

**Note:** The DTD syntax for declaring an element provides a way to specify a cardinality (a number of occurrences) for usage of elements in the content model.

# XML Schema

XML Schema:

- Is an XML language that defines and validates the structure of XML documents
- Is stored in an XML Schema Document (XSD)
- Defines components, such as:
  - Simple types definitions
  - Complex type definitions
  - Element declarations
  - Attribute declarations
- Supports XML namespaces and built-in, simple, and complex data types










## XML Schema

The W3C XML Schema Definition Language is an XML language (or vocabulary) used in an XML Schema Document (XSD) for describing and constraining the content of XML documents. The XSD is used to validate the structure of an XML document.

An XML document, whose structure is based on the definitions in an XML Schema, is called an *instance document* of that XML Schema. The slide shows an XML Schema Document stored separately from the XML instance document that it describes and validates.

The introduction of the XML Schema allowed XML technology to represent data types in a standard format. The data types give a precise way of specifying the type of content that can be held in elements and attributes of an XML document. Using a document type definition (DTD) provides no mechanism for specifying data types in a way that a database user may require. The XML Schema definition file builds upon DTD functionality while providing XML namespace and data type support.

## XML Schema (continued)

### Benefits of XML Schemas

You can use XML Schema to specify the structure of XML documents and validate the XML documents. Although most XML documents are well formed, they may not be "valid." A valid document is well formed and conforms to specific rules defined by either a DTD or an XML Schema.

Using XML Schema for validation has the following benefits. An XML Schema:

- Is easily created using XML, as defined by the W3C XML Schema language
- Supports the W3C namespace recommendation
- Can be used to validate content of text elements based on built-in and user-defined data types
- Allow the creation of complex data type models that can be reused
- Support object inheritance and substitution in types

In the Web publishing world, a single page that displays local weather, stock quotes, horoscopes, and specific news channels based on user preferences can involve dozens of queries made to underlying databases and application servers. These queries are made via SQL, the standard object-relational query language, or via a programmatic interface that ultimately calls SQL. Because both SQL and programming languages (such as Java) are strongly typed, they return information that possesses type, structure, constraints, relationships, and so on. The structural and data typing aspects of XML Schema can help exploit generation of viewable documents from databases.

# XML Schema Document: Example

- The simple XML Schema uses:
  - A required XML namespace string, with an `xs` prefix, `http://www.w3.org/2001/XMLSchema`
  - The `<schema>` element as its document root
  - The `<element>` element to declare an element

```
<?xml version="1.0"?>
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="departments" type="xs:string"/>
</xs:schema>
```

- A valid XML instance document:

```
<?xml version="1.0"?>
<!-- The element cannot contain child elements -->
<departments>
Finance
</departments>
```







### XML Schema Document: Example

The example in the slide shows a simple XML Schema document that declares a single element called `departments`. The XML Schema Document (XSD) uses the required XML namespace value `http://www.w3.org/2001/XMLSchema`, which is assigned the `xsd` namespace prefix. The `xsd` prefix is used to qualify the `schema`, `element`, and `string` names to ensure that the XSD document structure conforms to the W3C XML Schema language recommendation; that is, the XSD itself is a valid document.

The XML Schema language elements used in the example are the following:

- The `<xs:schema>` element is the root element for the XSD, and contains definitions for the structure of the XML instance document.
- The `<xs:element>` element declares a root element name, `departments`, for the XML instance document, as shown in the slide.
- The `<xs:string>` value is set as the data type for the contents of the `departments` element in the XML instance document. In this case, the data can be any string but not markup; that is, no child elements are permitted.

**Note:** Using a namespace prefix, such as `xs` or `xsd`, is recommended but not required.

# XML Schema Versus DTD

- XML Schema:
  - Is more powerful and flexible than a DTD
  - Provides better namespace support than a DTD
  - Is written in XML syntax
  - Is extensible
  - Provides data-type support
- DTD:
  - Provides the `ENTITY` functionality that is not supported by XML Schema
  - Can be embedded in an XML document
  - Is written in Standard Generalized Markup Language (SGML)







## XML Schema Versus DTD

An XML Schema defines a set of new names (called a *vocabulary*) and the structure and data types for elements, types, attributes, and attribute groups. A DTD defines a vocabulary of names for elements and attributes containing either a text string, or a combination of text strings and child elements.

Applications that process the same XML documents may want to represent their data elements differently. Using XML namespaces help applications to distinguish between different definitions of the same element name used in different contexts in a single document. XML Schema seamlessly supports namespaces to distinguish definitions. A DTD does not directly support namespaces or provide support for data typing. Because a DTD declares names associated with textual contents, it poses problems when you try to model dynamic content that comes from strongly typed systems such as databases. XML Schema is ideal for database systems because it provides for data type integrity and the maintenance of bound conditions.

A DTD is written using a syntax that is different from an XML document. Consequently, parsers, tools, and programs dealing with a DTD must implement the specific rules that govern the DTD. An XML Schema allows models to be written by using the XML syntax, which allows the same programs that read the data to also read the definition of the data.

# XML Path Language

XML Path Language (XPath):

- Is primarily used to address the nodes of an XML document modeled as a tree of nodes
- Is named after its use of a path notation for navigating through the hierarchical structure of an XML document
- Uses a compact, non-XML syntax to form expressions for use in Uniform Resource Identifier (URI) and XML attribute values
- Fully supports XML namespaces
- Is designed to be used by XML applications, such as XSLT and XPointer







## XML Path Language

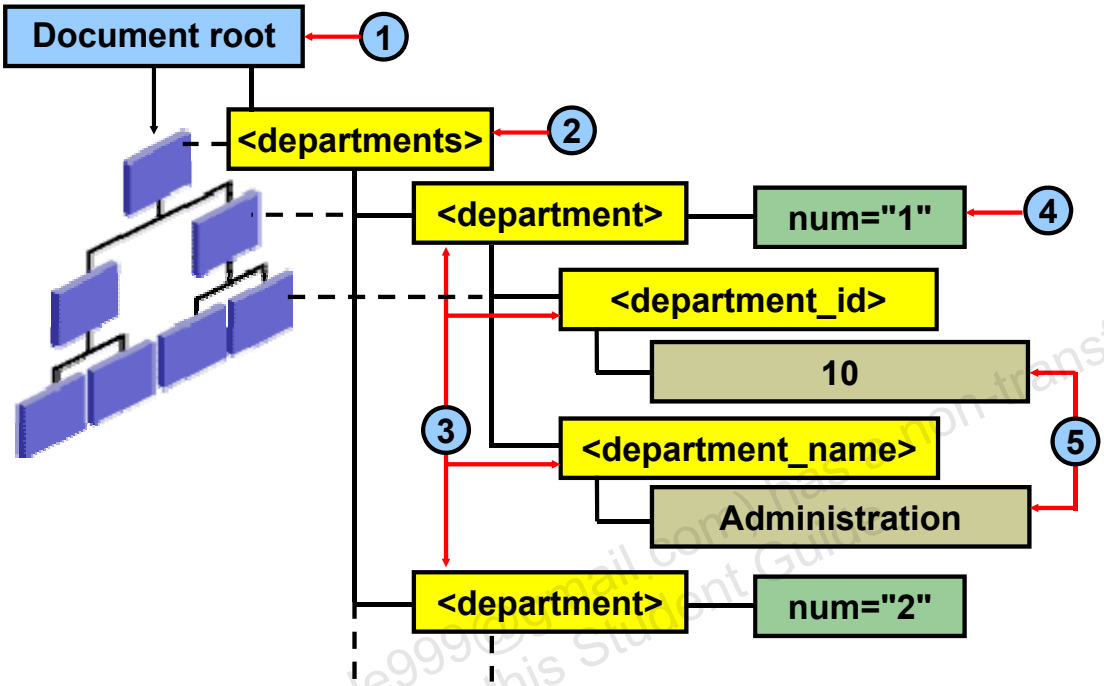
XML Path Language (XPath) is primarily used for addressing parts (nodes) of an XML document. XPath models an XML document as a tree of nodes and expresses a navigation path through the hierarchical structure of an XML document. XPath:

- Is named after the path notation that it uses for navigating through the structure of an XML document
- Uses a compact, non-XML syntax to form expressions that are used in URI and XML attribute values
- Facilitates the manipulation of string, number, and Boolean values
- Operates on the abstract, logical structure of an XML document called the document data model, rather than its surface syntax

In addition to its use for addressing parts of an XML document, XPath fully supports XML namespaces and provides a natural expression language subset for pattern matching, which is extensively used in the W3C XSLT Recommendation and in XPointer.

**Note:** XSLT is Extensible Stylesheet Language Transformation, the language used to transform an XML document into another XML document. XPointer is the XML Pointer Language used as a fragment identifier for any URI reference that locates a resource with a `MIME` type of `text/xml` or `application/xml`.

# XPath Model










## XPath Model

XPath models an XML document as a tree of nodes. The graphic in the slide depicts the different types of nodes, as seen by XPath, in an XML document:

1. Document root, which is a virtual document root that is the parent of the entire XML document containing the XML declaration, the root element and its children, comments, and processing instructions at the beginning and the end of the document
2. Root element (for example, the `<departments>` element).
3. Element nodes (for example, the `<department>`, `<department_id>`, and `<department_name>` elements. The root element is also an element node.
4. Attribute nodes (for example, the `num="1"` node)
5. Text nodes (for example, the nodes containing the text `10` and `Administration`)

**Note:** In the XPath model, the document root is not the same as the root element node.

XPath can also address comment nodes, processing instruction nodes, and namespace nodes, which are not shown in the diagram. XPath defines a way to compute a string value for each type of node, some of which have names. Because XPath fully supports the XML Namespaces Recommendation, the node name is modeled as a pair known as the *expanded name*, which consists of a *local part* and a *namespace URL*, which may be `null`.

XPath allows various kinds of expressions to be nested with full generality. XPath is a strongly typed language; this means that the operands of various expressions, operators, and functions must conform to designated types. XPath is a case-sensitive expression language.

# XSLT and XPath

XSLT:

- Transforms XML into plain text, HTML, or XML
- Specifies transformation rules in elements with attributes that use XPath expressions

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="//department_name">
    <html>
      <body>
        <p><xsl:value-of select="."/></p>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="*/text()"/>
</xsl:stylesheet>
```

ORACLE







## XSLT and XPath

An Extensible Stylesheet Language (XSL) document is an XML document with elements that define transformation rules. The slide shows sample XSL document elements:

- The `<xsl:template>` element with a `match` attribute containing an XPath expression defining the set of nodes to which the template rule applies.
- The `<xsl:value-of>` element with the `select` attribute containing an XPath expression. The `<xsl:value-of>` element outputs the text value of the specified node, relative to the context node matching its template XPath expression.

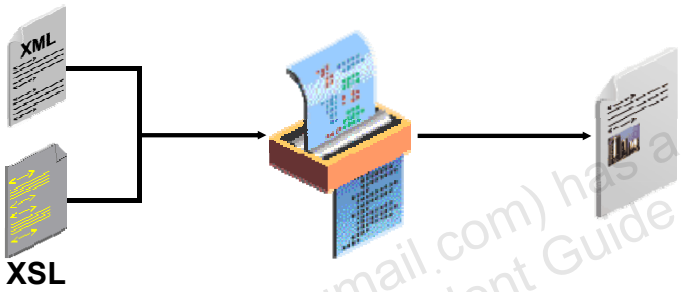
The `<xsl:template>` element informs the XSLT Processor how to locate specific element nodes in the input document, and provides the rules to create the output data as the contents specified between the start and end tags of the `<xsl:template>` element. In the example, you search for `department_name` elements, and use the current node (`.`) to output the value of the `department_name`.

**Note:** The XSLT Specification states that XSLT processors must provide a default template rule causing text nodes to be output, if no matching template rules exist. Suppress the output of unprocessed text nodes by including, at the end of the XSLT style sheet, a template rule matching all text nodes that does not output data. For example, as shown in the slide, you would use `<xsl:template match="*/text()"/>`.

# XSL

Extensible Stylesheet Language (XSL) has two parts:

- XSL Transformations (XSLT)
- XSL Formatting Objects (XSL-FO)










## XSL

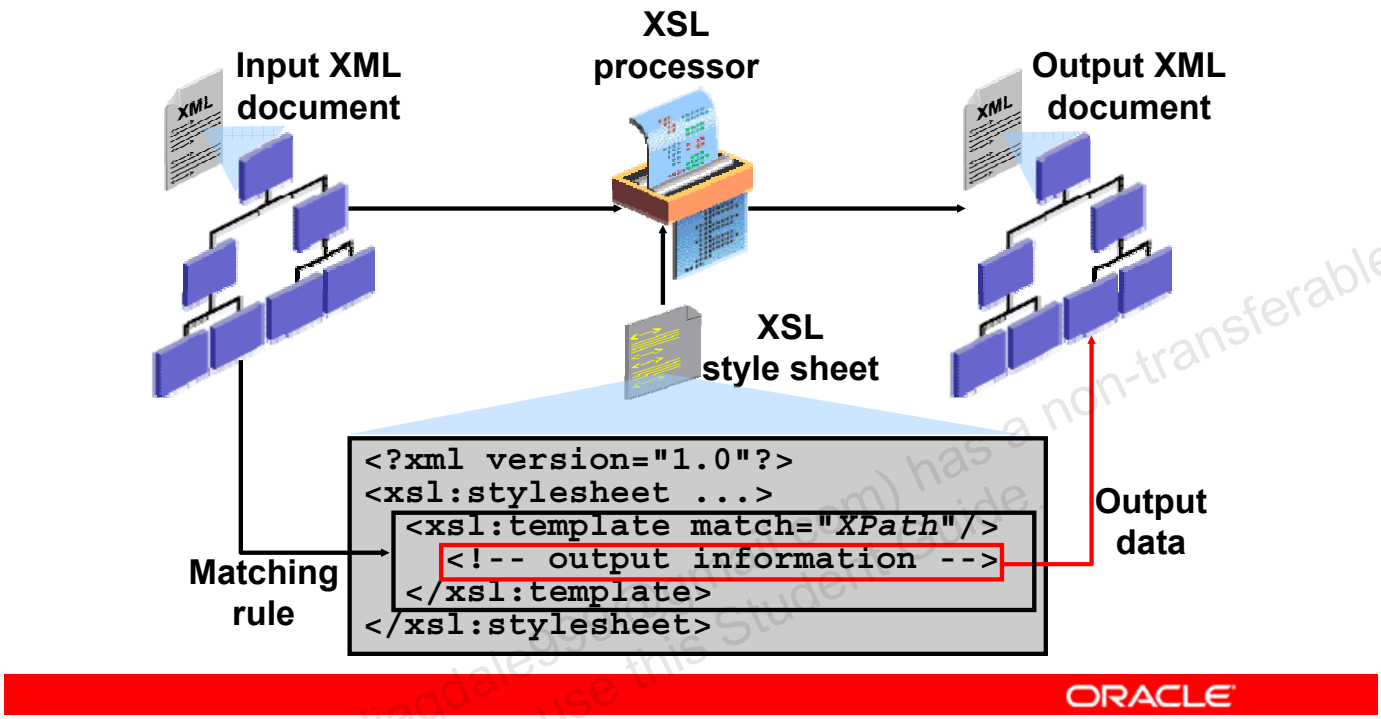
XML documents have structure and, unlike HTML, do not provide formatting or display instructions. Extensible Stylesheet Language (XSL) adds the capability of transforming an XML document into another document containing XML with a different structure, or an HTML document with formatting tags.

XSL has the following two parts:

- XSL Transformations (XSLT) is an XML application that processes rules contained in an XSLT style sheet. The rules in an XSLT style sheet can be applied to any XML document.
- XSL Formatting Objects (XSL-FO) is an XML application used for describing the precise layout of text on a page. XSLT is used to generate XSL-FO elements that are output to another XML document. Additional processing is needed to create the final output form, such as a portable document format (PDF) document.

The graphic in the slide depicts an XML document and an XSLT style sheet document being processed by an XSL Processor. The XSL Processor is an XML application that applies transformation rules to the XML document as specified in the XSL document, and produces a new document as the result.

# XSLT










## XSLT

Extensible Stylesheet Language Transformations (XSLT) is an XML application, or XSL Processor, that processes rules specifying how to transform one XML document into another XML document. The rules are contained in an XSLT document called an XSLT style sheet, which itself is an XML document and must be well formed.

The slide shows that an XSLT style sheet contains one or more template rules; each template rule has two parts:

- A match pattern, specified as an XPath expression in an attribute of an `<xsl:template>` element.
- Template data, appearing between the start and end tags of the `<xsl:template>` element, that contains the output information. The template data typically contains XML or HTML elements mixed with the XSLT rules.

The XSL Processor compares the elements in the input XML document with the template rule pattern in the XSLT style sheet, and writes the template data for matching rules to the output tree—typically, another XML document. Essentially, the transformation process operates on the tree data structure of a source (input) XML document and produces a tree of nodes as its output.

**Note:** XSL-FO elements are output elements found inside the `<xsl:template>` rules.

# XSLT Style Sheet

An XSLT style sheet is an XML document containing:

- A `<xsl:stylesheet>` root element declaring:
  - The `xsl` namespace prefix
  - The `http://www.w3.org/1999/XSL/Transform` mandatory namespace URI
- One or more `<xsl:template>` elements and other XSL elements defining transformation rules

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   ...
  <xsl:template match="/"> ... </xsl:template>
  <xsl:template match="..."> ... </xsl:template>
<xsl:stylesheet>
```







## XSLT Style Sheet

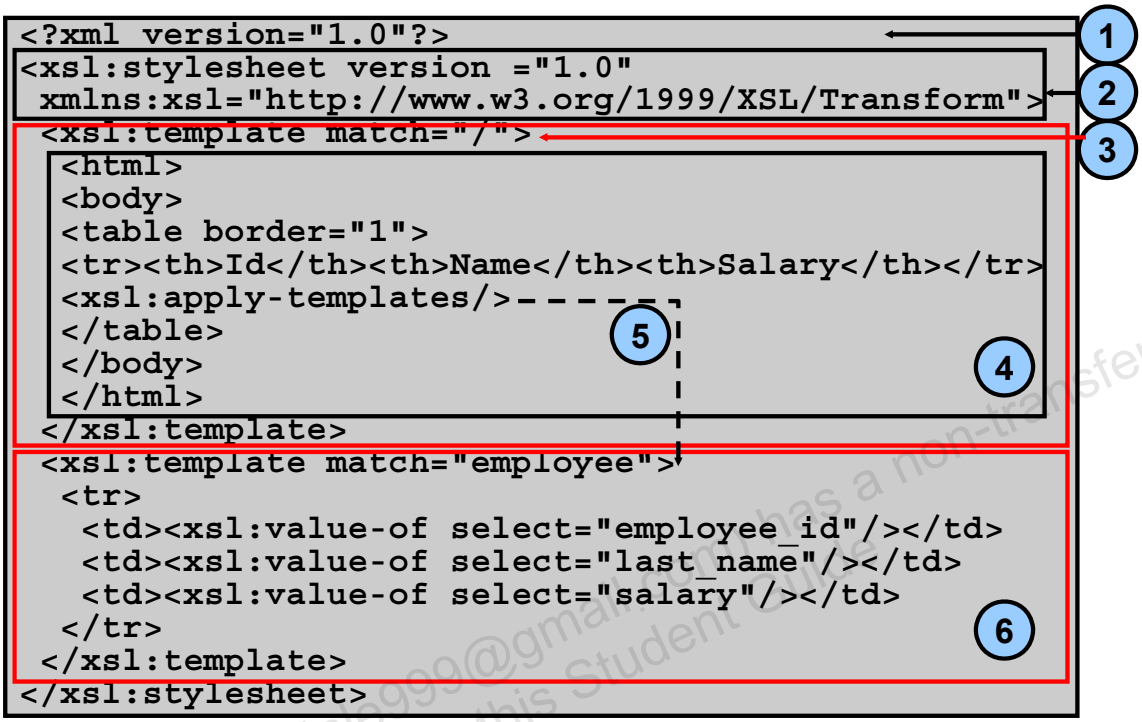
An XSLT style sheet is an XML document that uses elements from the XSLT vocabulary to describe transformation rules. The document element of every XSLT style sheet is the `<xsl:stylesheet>` element, whose content is a set of one or more XSL elements defining template rules describing the transformation to be performed.

**Note:** `<xsl:transform>` is a synonym for `<xsl:stylesheet>`.

The `<xsl:stylesheet>` element declares the mandatory namespace attribute `xmlns:xsl="http://www.w3.org/1999/XSL/Transform"` that is used to qualify the elements in an XSLT style sheet. If you do not provide this exact namespace prefix and URI, the XSLT Processor simply ignores the rules in `<xsl:template>`, `<xsl:for-each>`, `<xsl:value-of>`, and other XSL elements that are qualified with the `xsl` prefix. Therefore, it does not recognize them as XSLT instructions.

Each style sheet rule, called a *template*, contains a match pattern specified as an XPath expression that is compared against the nodes in the source XML document. An XSL template rule is represented by an `<xsl:template>` element with a `match` attribute that is assigned a `"pattern"` string containing an XPath expression.

# XSLT Style Sheet: Example










## XSLT Style Sheet: Example

The XSLT document in the example, called a *style sheet*, shows the following components:

1. The standard XML document declaration, because the style sheet is an XML document.
2. Mandatory XML namespace declaration using the `xsl` namespace prefix, and the URI `http://www.w3.org/1999/XSL/Transform` namespace.
3. The `<xsl:template>` rule matching the root of the source XML document.
4. The template data containing HTML tags to be written to the output document.
5. The `<xsl:apply-templates>` element instructing the XSLT Processor to apply template rules, if any, for child elements of root in the source document. The `<xsl:apply-templates>` element is replaced by any output generated for matching child element template rules.
6. The `<xsl:template>` rule for matching `<employee>` elements in the source document, with template data containing HTML table row (`<tr>...</tr>`) tags enclosing three table data (`<td>...</td>`) tag pairs. Each table data tag pair encloses an `<xsl:value-of select="..."/>` rule that inserts the value of the source document element selected by the XPath expression, which is typically the name of the source document element.

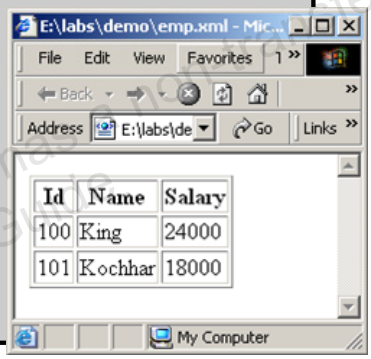
The XML document that uses this XSLT style sheet is shown in the slide titled “Viewing the Transformed Document.”

# Viewing the Transformed Document

Perform one of the following actions:

- Open the XML document in the browser.
- View the `oraxsl` command-line processor output.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="emp.xsl"?>
<employees>
  <employee>
    <employee_id>100</employee_id>
    <last_name>King</last_name>
    <salary>24000</salary>
  </employee>
  <employee>
    <employee_id>101</employee_id>
    <last_name>Kochhar</last_name>
    <salary>18000</salary>
  </employee>
</employees>
```






## Viewing the Transformed Document

The slide shows the source XML document and the result of applying the XSLT style sheet from the slide titled "XSLT Style Sheet: Example." When you open the XML document with Microsoft Internet Explorer, it automatically applies the XSLT style sheet specified in the XML document processing the instruction and renders the result shown.

Alternatively, you can use the `oraxsl` command-line interface to apply an XSLT style sheet to an XML document. To invoke `oraxsl`, use the following:

```
set CLASSPATH=E:\JDeveloper\lib\xmlparserv2.jar
java oracle.xml.parser.v2.oraxsl f.xml f.xsl f.html
```

The `oraxsl` parameters (in order) are:

1. The source XML document (`f.xml`).
2. The XSLT style sheet document (`f.xsl`).
3. The output file name (`f.html`). If omitted, the output appears in the standard output.

# Summary

In this lesson, you should have learned how to:

- Describe XML, XSL, and other X Standards objects
- List the components of an XML document
- Create a well-formed XML document
- Describe XML namespaces
- Describe a DTD
- Describe XML Schema
- Describe XML Path Language (XPath)
- Describe XSL and XSLT

# Practice 3: Overview

This practice covers the following topics:

- Creating an employee XML file
- Adding an employee element to the XML file
- Adding employee data to the XML file
- Creating a department XML file
- Adding a department element to the XML file
- Adding department data to the XML file
- Creating an XML file to combine employee and department

# 4

# Getting Started with BI Publisher



ORACLE

# Objectives

After completing this lesson, you should be able to do the following:

- Describe the sample schemas used in course practices (HR,OE, and SH)

# Sample Schemas in Oracle Database

Sample schemas:

- Are provided with Oracle Database
- Provide a common platform for examples in each release of Oracle Database
- Are a set of interlinked schemas







## Sample Schemas in Oracle Database

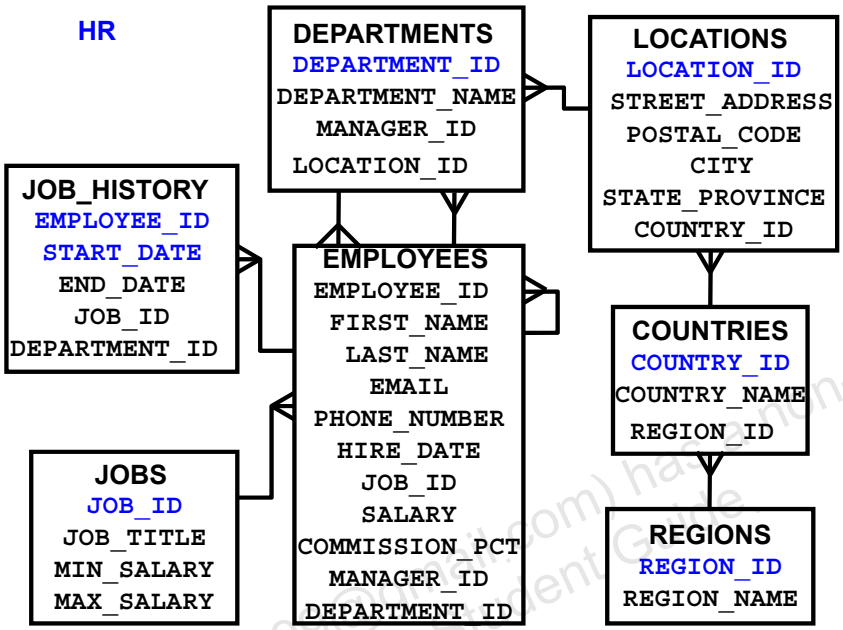
Oracle Database sample schemas provide a common platform for examples in each release of Oracle Database. Most of the examples in Oracle Database documentation, courses, and other training materials use the sample-schemas environment. These are a set of interlinked schemas that include a simple Human Resources (HR) schema, the Order Entry (OE) schema (of intermediate complexity), the Online Catalog (OC) subschema of the OE schema, the Product Media (PM) schema (dedicated to multimedia data types), the Information Exchange (IX) schema (that can demonstrate Oracle Advanced Queuing capabilities), and the Sales History (SH) schema (that is designed to allow for demos with large amounts of data).

In this course, HR, OE, and SH schemas are used for examples and in some practice sessions. These schemas are covered on the following pages.

**Note**

- Sample schemas are provided with Oracle Database. These can be installed automatically or manually depending on the installation options that you select. You can also install the required schemas by using the advanced or custom installation options.
- These schemas contain data about a global electronics retailer (a fictitious company), which sells several categories and subcategories of products through various channels.

# Human Resources (HR) Data Model





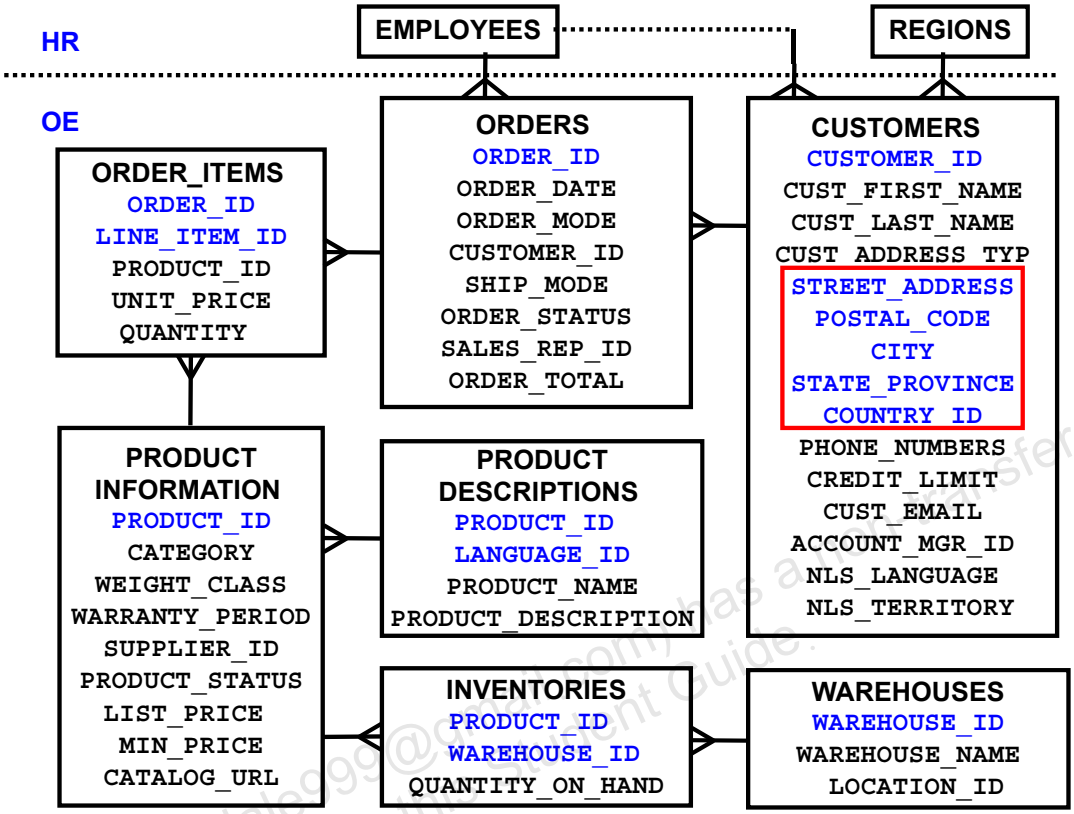

## Human Resources (HR) Data Model

The company's employee database is maintained in the Human Resource (HR) (schema) records. For example, each employee has an identification number, email address, job identification code, salary, and manager. Some employees earn commissions in addition to their salary. The company also tracks information about jobs within the organization. Each job has an identification code, job title, and a minimum and maximum salary range for the job. Some employees have been with the company for a long time and have held different positions within the company. When an employee resigns, the duration of the employment, the job identification number, and the department are recorded.

The sample company is regionally diverse, so it tracks the locations of its warehouses and departments. Each employee is assigned to a department, and each department is identified either by a unique department number or a short name. Each department is associated with one location, and each location has a full address that includes the street name, postal code, city, state or province, and the country code.

In places where the departments and warehouses are located, the company records details such as the country name, currency symbol, currency name, and the region where the country is geographically.

# Order Entry (OE) Data Model






## Order Entry (OE) Data Model

As shown in the slide, the OE schema is interlinked with the HR schema. The OE schema contains data about orders, products, customers, inventory information, and so on. The company maintains information about products, such as product identification numbers, the category into which the product falls, order entry (OE), the weight group (for shipping purposes), the warranty period if applicable, the supplier, the availability status of the product, a list price, a minimum price at which a product is sold, and a URL address for manufacturer information. Inventory information is also recorded for all products, including the warehouse where the product is available and the quantity on hand. Because products are sold worldwide, the company maintains the names of the products and their descriptions in several languages.

The company maintains warehouses in several locations to fulfill customer needs. Each warehouse has a warehouse identification number, name, facility description, and location identification number.

Customer information is also tracked, which contains customer identification number, name, street name, city or province, country, phone numbers (up to five phone numbers for each customer), and postal code. Some customers place orders through the Internet, so email addresses are also recorded. Because of language differences among customers, the company records the native language and territory of each customer. The company places a credit limit on its customers to limit the amount of products that they can purchase at one time. Some customers have an account manager, and this information is also recorded.

## Order Entry (OE) Data Model (continued)

When a customer places an order, the company tracks the date of the order, how the order was placed, the current status of the order, shipping mode, total amount of the order, and the sales representative who helped place the order. The sales representative may or may not be the same person as the account manager for a customer. If an order is placed over the Internet, no sales representative is recorded. In addition to the order information, the company also tracks the number of items ordered, the unit price, and the products ordered.

# Summary

In this lesson, you should have learned how to:

- Describe the sample schemas used in course practices (HR,OE, and SH)

# 5

# Creating Simple RTF Templates

# Objectives

After completing this lesson, you should be able to:

- Describe the functions and features of BI Publisher Desktop
- Describe how to install BI Publisher Desktop
- Create RTF templates for a sample report
- Create and publish RTF templates for BI Publisher reports
- Create RTF templates by using the following methods:
  - Basic
  - Form field
- Insert tables,forms, and charts in RTF templates
- Preview results



**Note**

It is worth noting that Oracle BI Publisher Desktop was previously known as “Template Builder for Word” or just as “Template Builder.” In this course, these terms are used synonymously.

# Introduction to Oracle BI Publisher Desktop

What is Oracle BI Publisher Desktop?

- An extension to Microsoft Word (2000 or later) that simplifies rich text format (RTF) template creation

Oracle BI Publisher Desktop enables you to perform the following tasks:

- Inserting data fields
- Inserting data-driven tables
- Inserting data-driven forms
- Inserting data-driven charts
- Previewing your template with sample XML data
- Browsing and updating the content of form fields
- Extracting boilerplate text into an XML Localization Interchange File Format (XLIFF) translation file and testing translations







**Introduction to Oracle BI Publisher Desktop**

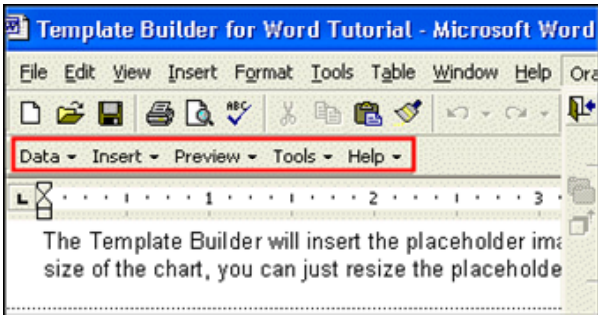
Rich Text Format (RTF) is a specification used by common word processing applications such as Microsoft Word. When you save a document, RTF is a file type option that you select. Oracle BI Publisher Desktop (previously known as Template Builder for Word) is an extension to Word, and simplifies the common tasks associated with creating RTF templates for Oracle BI Publisher reports.

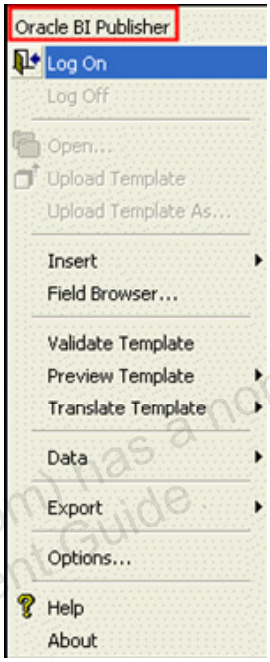
You can create report designs by using many design features of your standard Word application, and BI Publisher's RTF template parser will recognize and maintain the design. In addition to Word's formatting features, BI Publisher Desktop supports other advanced reporting features such as charts, conditional formatting, and running totals. BI Publisher Desktop provides easy-to-use wizards for inserting fields, tables/forms, charts, and cross-tabs. BI Publisher Desktop also enables you to preview reports by using the template in supported formats. It also supports translation features.

# BI Publisher Desktop User Interface (UI)

Oracle BI Publisher Desktop UI consists of the following:

- BI Publisher menu
- BI Publisher toolbar










## BI Publisher Desktop User Interface (UI)

After you install BI Publisher Desktop successfully, when you open the Word application, you see the BI Publisher menu and the BI Publisher toolbar (menu bar) with various menus such as Data, Insert, Preview, Tools, and Help. These menus are explained in detail on the following pages.

### BI Publisher Menu

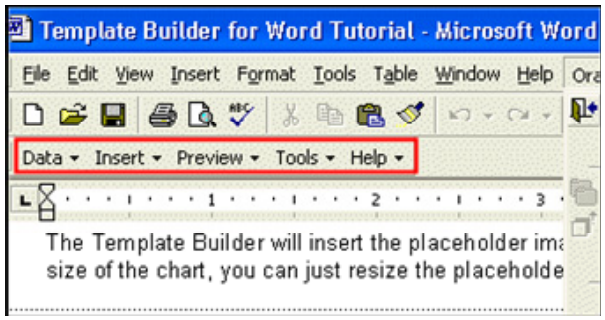
The Oracle BI Publisher menu includes functions that enable you to log on and interact with the BI Publisher Server; validate, preview, and publish templates; load data; insert various objects; and so on. A few important options are discussed here:

- **Log On:** This invokes the BI Publisher logon page. Enter your username and password. Select or enter the URL for the BI Publisher Report server. When you log on, the Open Template dialog box is displayed.
  **Note:** You must be granted one of the following roles to log on through BI Publisher Desktop: BI Publisher Administrator, BI Publisher Developer, or BI Publisher Template Designer.
- **Open Template:** This invokes the Open Template dialog box. This dialog box enables you to select reports from the BI Publisher Report server to create RTF templates for these reports.

## BI Publisher Desktop User Interface (UI) (continued)

- **Update Layout Template:** If you used the Open Template dialog box to download a template from the BI Publisher Server, use this option to upload the updated layout back to the report definition on the server.
- **Publish Template As:** If you used the Open Template dialog box to download a template or to open a report from the BI Publisher Server, use this option to upload the layout to the report definition on the server. Also use this option to upload modifications to an existing template under a different name.

# BI Publisher Toolbar









## BI Publisher Toolbar

The toolbar or menu bar gives you alternative access to much of the functionality provided by the Oracle BI Publisher menu. Data, Preview, and Help menus are discussed here; Insert and Tools are discussed on the following pages.

**Data Menu**

From the Data menu, you can select the data source for the template. The data source defines the XML format that is merged with the RTF template. You must load a data source to use most of the Template Builder functionality.

- **Load XML Data:** Using this option, you can load a sample XML file that contains all fields that you want to insert into your template as a data source. If you are not connected to the BI Publisher Server, use this method to load your data.
- **Load XML Schema:** With this option, you can load an XML Schema file (`.xsd`) that contains the fields available in your report XML data. The XML Schema has the advantage of being complete (a sample XML file may not contain all the fields from the data source). For the preview, the Template Builder can generate dummy sample data for an XML Schema. However, the preview works better if you also upload data from an actual sample.

## BI Publisher Toolbar (continued)

### Preview Menu

The Preview menu enables you to preview your RTF template with sample XML data. Use the Data menu to upload a sample XML file. The Preview menu offers you PDF, HTML, RTF, EXCEL, PowerPoint, and CSV as output formats. When you select any of these output formats, the Template Builder uses BI Publisher to merge the data into your template and create the output document.

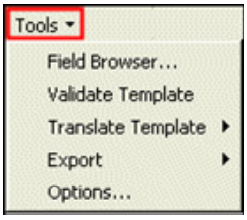
**Note:** You must have installed Adobe Acrobat Reader version 5.0 or later to preview documents in PDF format.

### Help Menu

Using the Help menu, you can access online Help. The "About" option gives information about the product, such as the copyright notice and the version of Oracle BI Publisher Desktop.

# BI Publisher Toolbar: Tools

- Field Browser
- Validate Template
- Translate Template
- Export
- Options







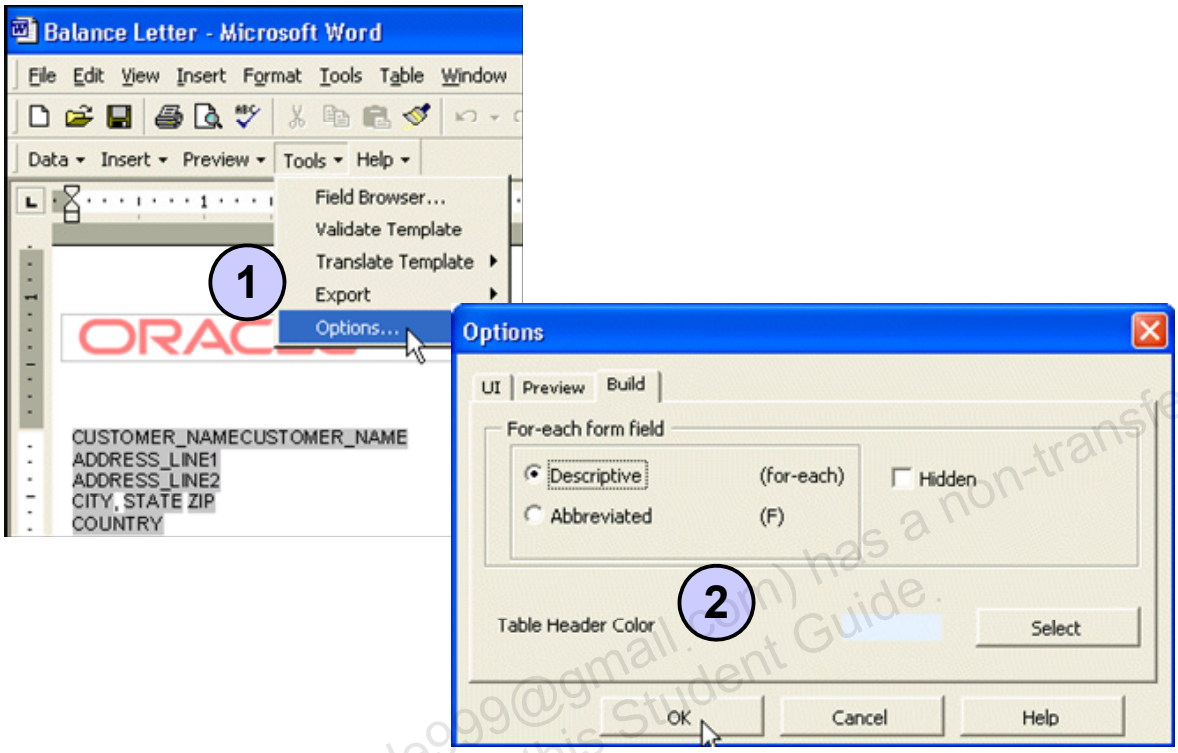

## BI Publisher Desktop: Tools

### Tools Menu

The Tools menu provides you with additional tools that help you translate and modify RTF templates for Oracle BI Publisher.

- **Field Browser:** The field browser is a tool for advanced users who need to change the BI Publisher commands hidden in the form fields. It shows the commands behind each form field and allows you to change them. Use this tool to correct flawed RTF templates or to update multiple fields efficiently.
- **Validate Template:** The validation function checks the template for incorrect use of BI Publisher commands and unsupported elements in the Word file.
- **Translate Template:**
  - **Extract Text:** This option enables you to create a standard XLIFF translation file containing the boilerplate text from your template. The XLIFF file is a standard file format that is understood by many translation software packages.
    Preview Translation: Using this option, you can preview your template as a PDF file by using a specified XLIFF translation file. This functionality enables you to test translation files.
  - **Localize Template:** This option applies a translation file to an RTF template. This means that, in your current RTF template, all boilerplate text is translated. The main function of this feature is to create a language-specific version of a template.
- **Export:** This enables you to convert RTF templates into an enhanced XSL-FO style sheet, FO-formatted XML, or PDF. This function can be used to generate XSL-FO for debugging or further customization.
- **Options:** This enables you to define some preferences and options for using BI Publisher.

# Setting Build Preferences



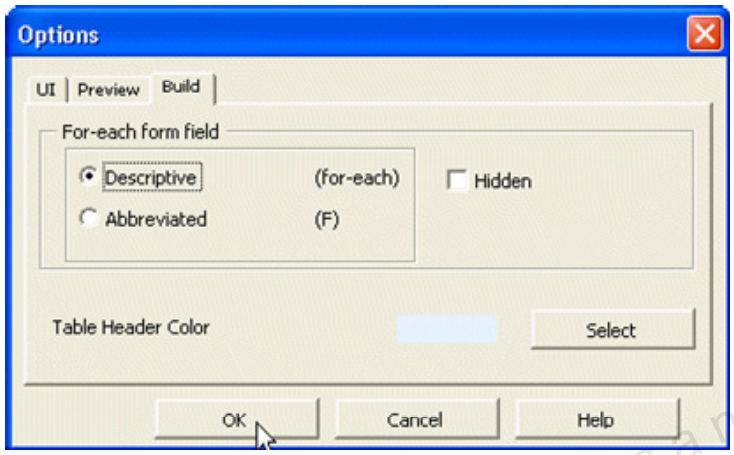



## Setting Build Preferences

You can determine how the Template Builder should generate tables and forms by setting build preferences:

1. In the Tools menu, select Options. (You can also select Oracle BI Publisher > Options.)
2. Click the Build tab. The options on this page enable you to specify how the tables and forms must be generated. After selecting the options, click OK.

# Build Options






ORACLE



## Build Options

The Build options are described here:

**For-each form field**

You may choose how BI Publisher Desktop creates form fields for processing instructions in the Insert Table/Form dialog box.

- The **Descriptive** option (for-each Invoice) renders a descriptive form field for processing instructions that makes the layout template easier to understand. However, the longer fields may be distorted in the visual layout of the template.
- The **Abbreviated** option (F) provides a one-letter abbreviation for each instruction.

If the **Hidden** font effect check box is selected, the instruction text is generated with the hidden font effect of Microsoft Word. The hidden text is not visible in the print preview. Also, you may display or hide the text by selecting or deselecting the **Hidden Text** check box on the **View** tabbed page of the Microsoft Word **Options** dialog box.

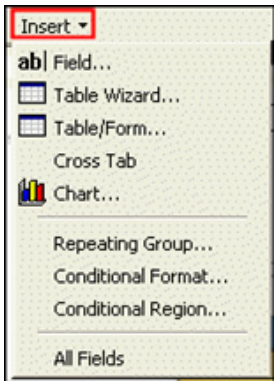
**Table Header Color**

You may also choose the color of the table header columns for the generated tables in the **Insert Table/Form** dialog box.

**Note:** You have inserted fields in the sample template. You can also insert tables and charts, and this is covered in more detail on the following pages.

# BI Publisher Toolbar: Wizards

- Table Wizard
- Table/Form
- Cross Tab
- Chart






ORACLE



## BI Publisher Toolbar: Wizards

BI Publisher also offers you easy-to-use, step-by-step wizards for inserting tables, forms, charts, and cross-tabs in your RTF templates. BI Publisher Desktop also provides you with easy-to-use wizards for creating conditional formats, conditional regions, and so on. Some of these wizards are discussed with examples in this lesson; the cross-tab wizard and conditional formats are discussed in the lesson titled "Advanced RTF Template Techniques." Most of these wizards are provided through the Insert menu as discussed here:

**Insert Menu**

Using the Insert menu, you can insert data fields and groups from the data source into the template:

- **Field:** This option enables you to select fields from your data source and insert them into your template.
- **Table Wizard:** This option provides you with a wizard that guides you through the creation of tables used in typical reports.
- **Table/Form:** This option enables you to insert tables or forms into your template. It opens the Insert Table/Form dialog box. From this dialog box, you can insert data fields to be organized as a simple or nested table or as a form that is repeated with different data. You may even organize all data fields for the whole document before inserting them.
- **Cross tab:** This option enables you to quickly insert a pivot table of your data. You can drag data elements to the cross-tab structure.

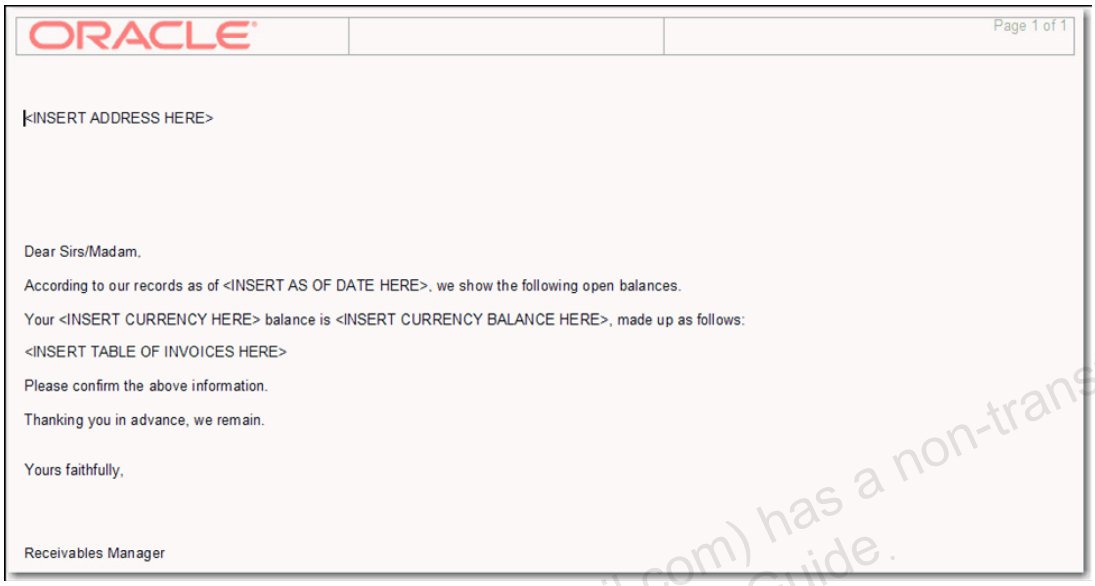
## BI Publisher Toolbar: Wizards (continued)

### Insert Menu (continued)

- **Chart:** BI Publisher does not recognize native Microsoft Word charts. With the Insert Chart option, you can insert a chart that is understood by Oracle BI Publisher. It opens the Insert Chart dialog box, where you can define parameters for your chart.
- **All Fields:** This option inserts all fields found in the XML data into your document. It also inserts processing instructions that repeat a section, such as a table row, when the associated XML element is repeated.
- Other options in the menu enable you to insert a conditional format, conditional region, and a repeating group.

**Note:** XML documents often contain a large number of fields in a deeply nested hierarchy. For example, an Oracle Purchasing purchase order contains purchase order lines, which contain shipments, which contain distributions. The purchase order line alone contains more than 150 data fields. In these cases, you must use the Insert Table/Form function to have more control over which fields are inserted.

# Creating an RTF Template from a Sample








## Creating an RTF Template from a Sample

A sample template file, `Balance Letter.rtf`, is shown in the slide. You often have a template from which to begin. Files used in this example and the associated lab files are found in the following location:

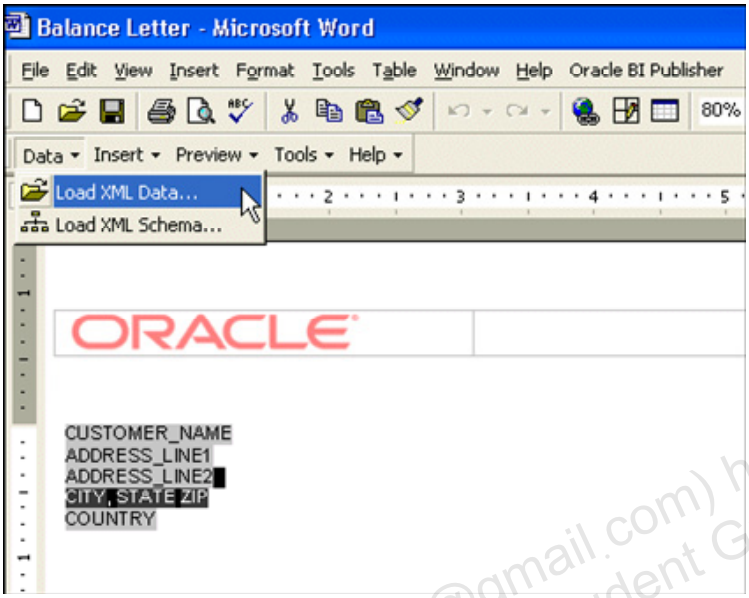
```
D:\Program Files\Oracle\BI Publisher\BI Publisher Desktop\samples\RTF templates\Balance Letter\
```

**Note:** The default location for sample files is:

```
D:\Program Files\Oracle\BI Publisher\BI Publisher Desktop\samples
```

Also, the template and lab files are provided in the D:\Labs\BIP_10.1.3.3._Labs folder for this course on your system.

# Creating an RTF Template: Loading the Sample XML Data





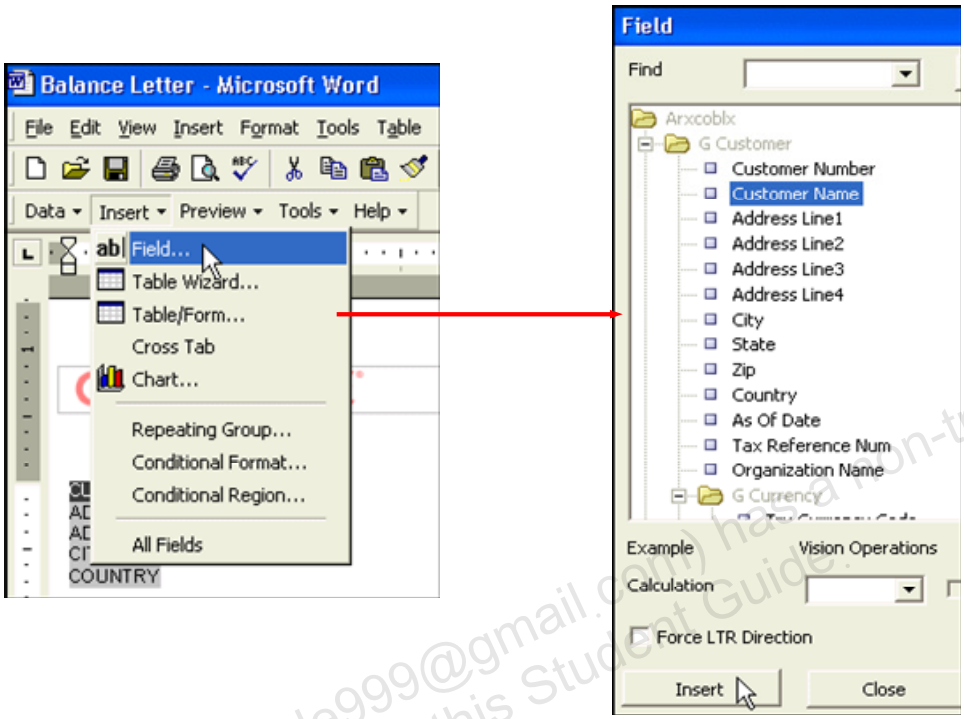


## Creating an RTF Template: Loading the Sample XML Data

1. Select Load XML Data from the Data menu.
2. Open the `Balance.xml` file as your XML sample data from the samples folder. When the XML data is loaded, a message indicating the successful loading of data is displayed.

**Note:** You can also open a BI Publisher report to load the data. You must keep the template open while you load the data.

# Creating an RTF Template: Inserting Fields






## Creating an RTF Template: Inserting Fields

Using the Insert Field feature, you can select data elements defined in the data source (that is, the XML file) and insert them into the template. Position the cursor in the Word document where you want to insert a field (for example, Customer Name). From the Insert menu, select Field to open the Field dialog box. The Field dialog box shows the structure of the data source in a tree view. Select <CUSTOMER_NAME> and click Insert.

Similarly, insert other fields: <ADDRESS_LINE_1>, <ADDRESS_LINE_2>, <ADDRESS_LINE_3>, <ADDRESS_LINE_4>, <CITY>, <STATE>, and <ZIP>

When you insert a field in a template, a text form field, with hidden BI Publisher commands (in the Help text of the form field), is inserted at the cursor position in the template.

You may either select and insert additional data fields or close the dialog box by clicking the Close button. Using the Insert Field dialog box, set the other fields in the RTF template, such as <AS_OF_DATE>, <TRX_CURRENCY_CODE> , and <C_INV_OPEN_BALANCE>.

## Creating an RTF Template: Inserting Fields (continued)

**Note**

- Throughout the same template file, you see placeholders, such as <INSERT ADDRESS HERE>. You must remove these placeholders, and replace them with appropriate fields.
- You must also adjust the position of the cursor, typically by pressing the Enter key, when inserting fields. Otherwise, all fields may appear on the same line.
- You can also drag the fields from the Field window into your template. At this point, you must insert fields only for unique data fields. You handle the data table in the next section.

# Creating an RTF Template: Previewing Data

In the Preview menu, select your format to view data.


This example shows the RTF format.



## Creating an RTF Template: Previewing Data

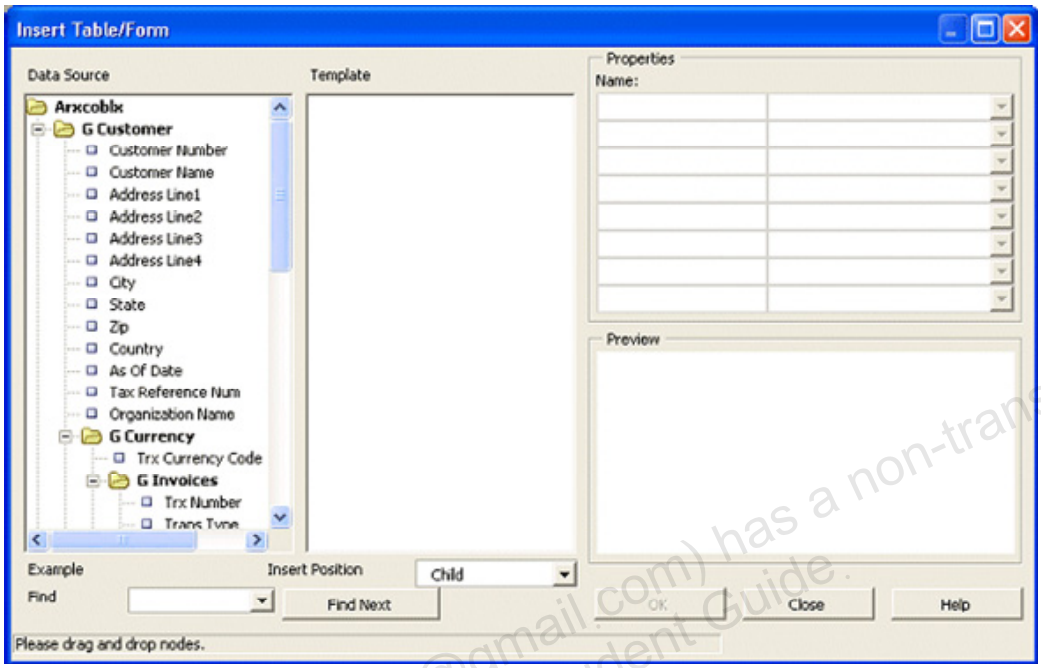
Select a format from the Preview menu. You can also select a format from BI Publisher> Preview Template.

Before previewing your work, be certain to perform a Save As instead of Save.

Your choice of preview formats are:

- PDF to see the document in Acrobat Reader
- HTML to see the document in your browser
- RTF to see the document in Microsoft Word
- EXCEL to see the document in Microsoft Excel

# Creating an RTF Template: Inserting a Table






## Creating an RTF Template: Inserting a Table

From the Insert menu, select Table/Form to open the Insert Table/Form dialog box. The Insert Advanced Table/Form dialog box is the most flexible tool of the Template Builder. It enables you to perform the following tasks:

- Create a simple or nested table with a variable number of rows
- Associate a group of data elements, such as a complete invoice or a purchase order line, with a form in the document that is repeated for each occurrence of the data element
- Select and define a layout for all data fields in the template
- Group or regroup the data

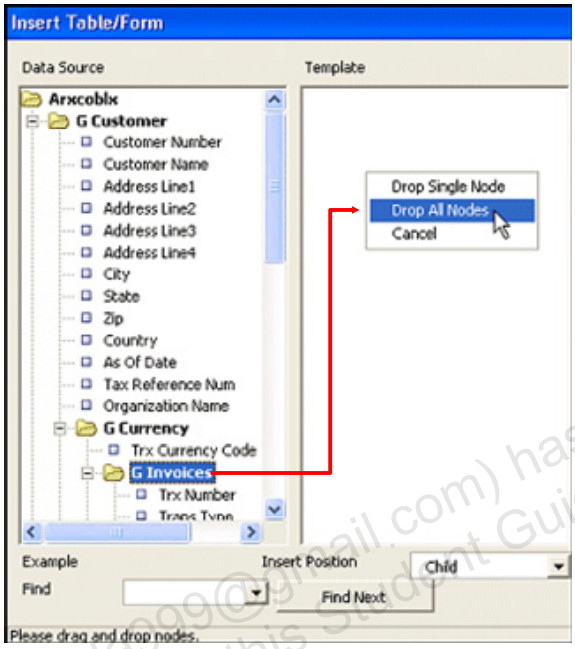
The Insert Table/Form dialog box displays two tree view panes. The left pane shows the data source structure, and the right pane shows the elements that were selected.

**Note:** Oracle BI Publisher Desktop generates the table by using Build Options settings that you specified previously. The following example uses descriptive instructions.

**Example:** The document must include a table of all invoices that are not paid or only partially paid. To open the Table/Form dialog box, select the <INSERT TABLE OF INVOICES HERE> label. Next, select Insert > Table/Form.

# Selecting Data Fields

Drag the data sources to the Template pane.






## Selecting Data Fields

First, select the data fields that you want to insert in the template and then define how to format them. Drag an XML element from the left Data Source pane to the right Template pane to select it. If the XML element has children, you will see a pop-up menu with the following options:

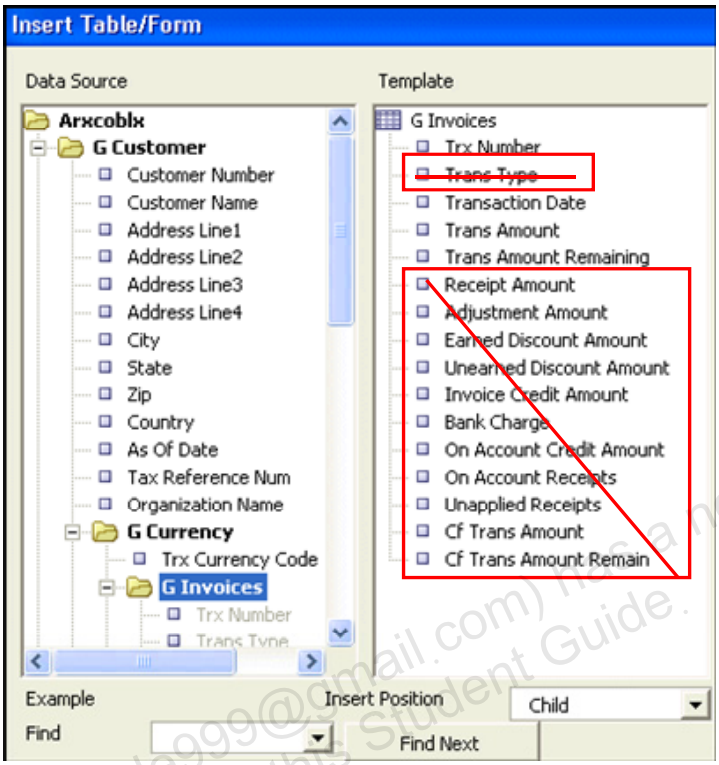
- **Drop Single Node:** Select this option if you want to move only the selected node.
- **Drop All Nodes:** Select this option if you want to move the node and all its children.
- **Cancel:** This cancels the selection.

If you drag an additional data field from the left Data Source pane to the right Template pane, it is either inserted at the same level **(**Same Level) or below the node (Child) where you release the node. The Insert Position box defines which way the node is inserted.

**Note:** If you use the left mouse button for the drag-and-drop operation, the node and all children are copied. However, if you use the right mouse button for dragging, a dialog box appears when you release the mouse button. The dialog box gives you the option to copy either only the selected node or the selected node and all the children.

**Example:** Drag G Invoices from the Data Source pane to the Template pane. Choose Drop All Nodes in the pop-up menu.

# Removing Selected Fields






## Removing Selected Fields

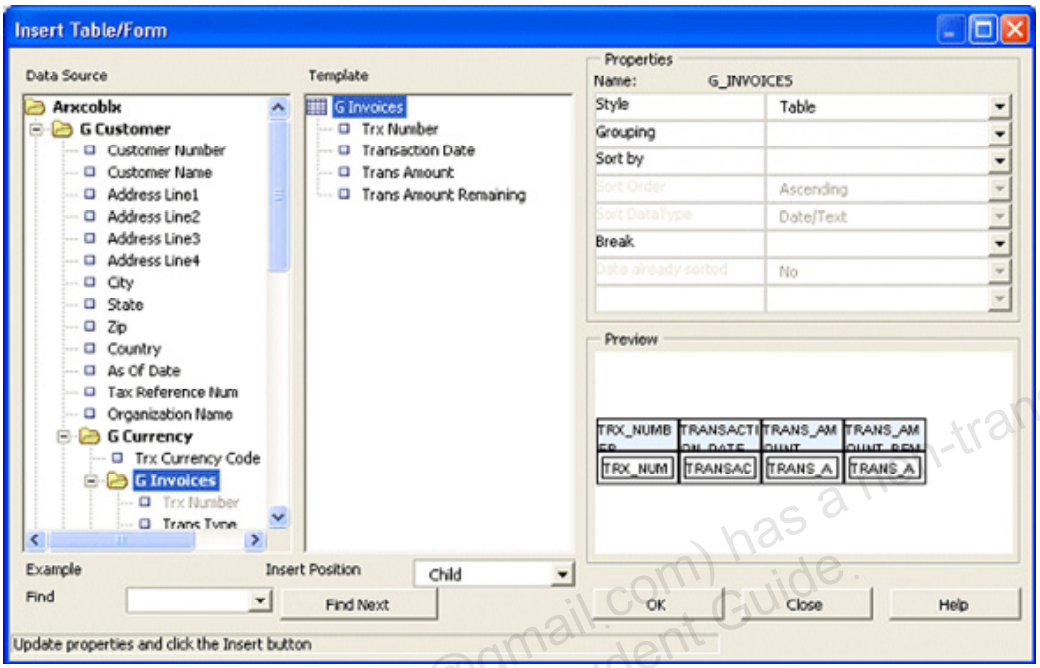
You can also remove fields from the Template pane by selecting a field and pressing the Delete key. Only one field may be deleted at a time.

**Example:** After selecting the Drop All Nodes option, G Invoices and all its child fields are added to the Template pane, as shown in the slide. Now, delete all the child fields except:

- Trx Number
- Transaction Date
- Trans Amount
- Trans Amount Remaining

Alternatively, you could have selected G Invoices, chosen Drop Single Node, and then dragged over just the children that you want to include in your table.

# Completing the Table






## Completing the Table

Click a parent node element in the Template pane to preview how the data fields are formatted. In the slide example, the G Invoices element is selected in the Template pane. The Properties and Preview regions on the right show how the data fields are formatted.

For each group, you can set the following properties that describe how a group (such as G Invoices) must be rendered by Oracle BI Publisher Desktop:

- **Style:** Select Table to create a table around the data fields.
- **Grouping:** Grouping is an advanced operation that enables you to regroup data—for example, by transaction date instead of currency. You can select the element that the data must be grouped by for this property. Refer to the *Oracle BI Publisher User's Guide* for additional information about grouping.
- **Show Grouping Value:** This property is shown only if you have selected a node created by the Grouping functionality. You can choose for a group if the Data Field node that is used as a grouping criterion is shown in the table or form.
- **Sort By:** You can select an element by which data groups are sorted.
- **Sort Order:** If you have selected an element for the Sort By operation, you can select if the data should be sorted in ascending or descending order.

## Completing the Table (continued)

- **Sort Data Type:** If you have selected an element for the Sort By operation, the data is sorted as Text by default. That means that 12 will be shown after 111. If you need to sort numbers, you must select Number as the sort data type.
- **Break:** This property enables you to insert a page break or a section break between every data group. If nothing is selected, the data groups are shown continuously with no break. If you select New Page per Element, a page break will be inserted between each element. If you select New Section per Element, a section break will be created for each data group. A section break allows changing the header/footer and resets the page number. You typically use this option if you want to print multiple documents (for example, invoices or purchase orders) to a single PDF file.

Click the OK button to create the table.

**Note:** The order in which the data elements are shown reflects the order of the columns in the table. If you want to reorder the columns, you must change the option in Insert Position from Child to Same Level. Then drag the elements into the correct order.

# Resulting Table

The resulting table should look like the following:

| Trx Number | Transaction Date | Trans Amount | Trans Amount Remaining |
|---|---|---|---|
| F TRX_NUMBER | TRANSACTION_DATE | TRANS_AMOUNT | TRANS_AMOUNT_REMAINING E |







## Resulting Table

Oracle BI Publisher Desktop creates two kinds of form fields:

- Form fields representing data elements
- Form fields with processing instructions for repeating table rows or a repeating document section

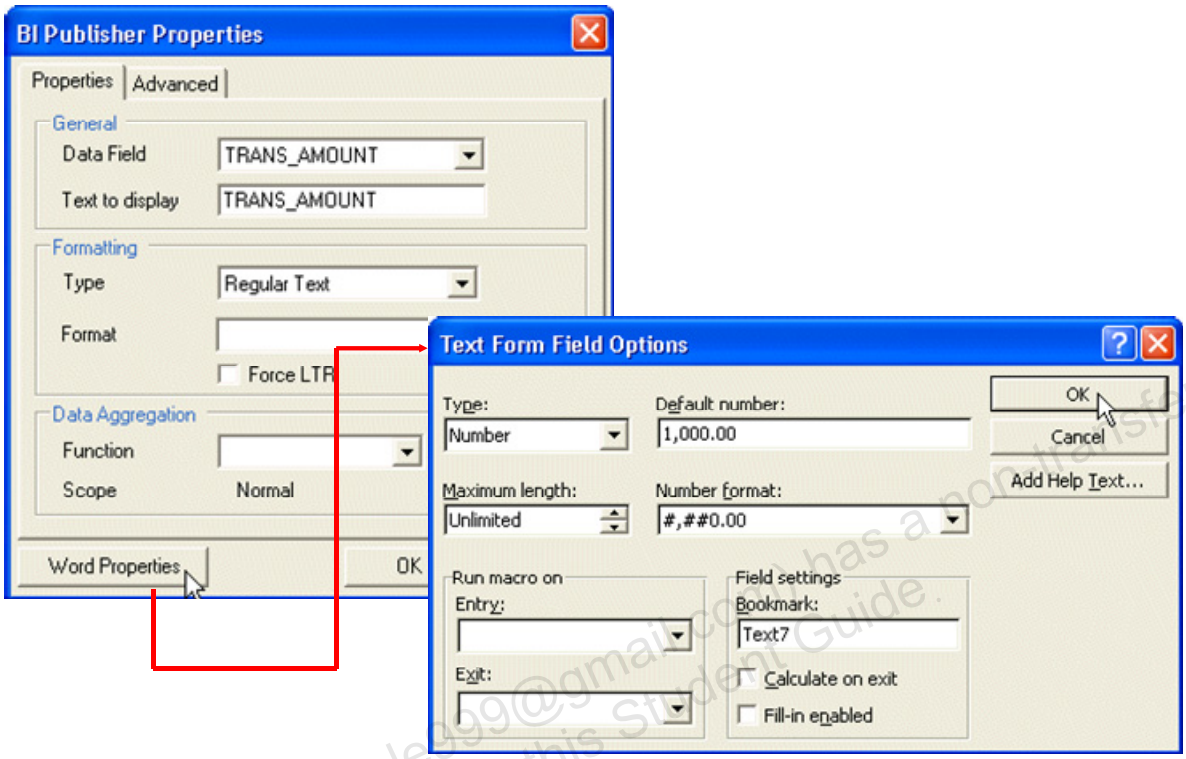
Form fields representing data elements are replaced with the data when the template is processed. The "**F** TRX_NUMBER" and "TRANS_AMOUNT_REMAINING **E**" form fields indicate a repeating section (F stands for for-each, and E for end). The section of the document encapsulated by these two form fields is repeated if the associated data element, G_INVOICE, is found repetitively in the data.

You can now format these elements in Microsoft Word and add more text. You can also move fields around as long as you keep them in the surrounding for-each G_INVOICES and end G_INVOICES processing instructions. If you remove one of these processing instructions, your template may not work with Oracle BI Publisher.

You can now preview the template again to review your current template (see the slide).

You probably want to change the alignment of the table columns or change the data type and format for the amounts. For example, select the TRANS_AMOUNT field, right-click, and select Properties to see the Text Form Field Options dialog box.

# Changing Field Properties





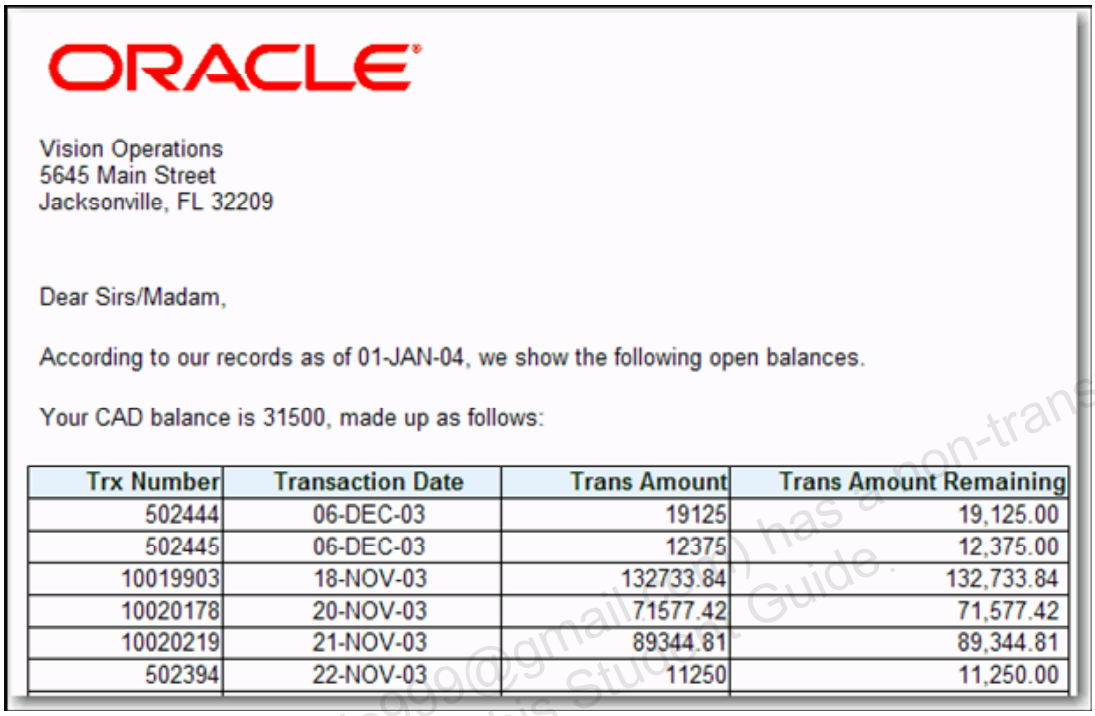

## Changing Field Properties

BI Publisher enables you to change the properties and format of the fields. Double-click a field to edit the properties (for example, the TRANS_AMOUNT field). This opens the BI Publisher Properties dialog box. (This dialog box can also be opened by using the shortcut menu of a field.) You can change the properties of the fields in Word. Click Word Properties to open the Form Field Properties dialog box and edit the format as follows: Select Number in the Type field, enter `#,##0.00` for the number format, and enter `1,000.00` as the default number to achieve a more desirable format for currencies in US dollars. Formatting options defined in this dialog box are understood by Oracle BI Publisher and used to format your fields. Try to modify the template to get an idea of which Word functions are supported by Oracle BI Publisher.

**Note:** It is recommended that you do not use currency signs such as $ in the format string, especially if you are using multiple languages with a single template or the translation, because it may lead to difficulties.

**Example:** Set the field properties shown in the slide to both the TRANS_AMOUNT and TRANS_AMOUNT_REMAINING fields. Then, in Word, set all columns except Transaction Date to display right-justified.

# Previewing the Table Data



ORACLE

Vision Operations
5645 Main Street
Jacksonville, FL 32209

Dear Sirs/Madam,

According to our records as of 01-JAN-04, we show the following open balances.

Your CAD balance is 31500, made up as follows:

| Trx Number | Transaction Date | Trans Amount | Trans Amount Remaining |
|---|---|---|---|
| 502444 | 06-DEC-03 | 19125 | 19,125.00 |
| 502445 | 06-DEC-03 | 12375 | 12,375.00 |
| 10019903 | 18-NOV-03 | 132733.84 | 132,733.84 |
| 10020178 | 20-NOV-03 | 71577.42 | 71,577.42 |
| 10020219 | 21-NOV-03 | 89344.81 | 89,344.81 |
| 502394 | 22-NOV-03 | 11250 | 11,250.00 |

ORACLE



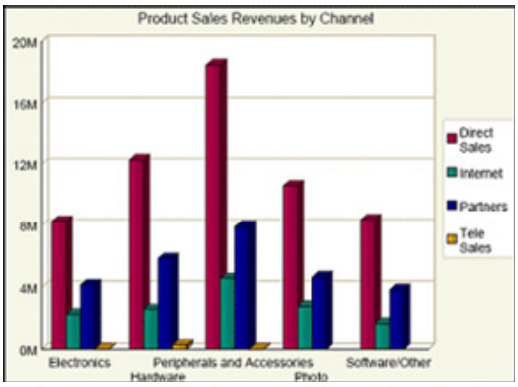
## Previewing the Table Data

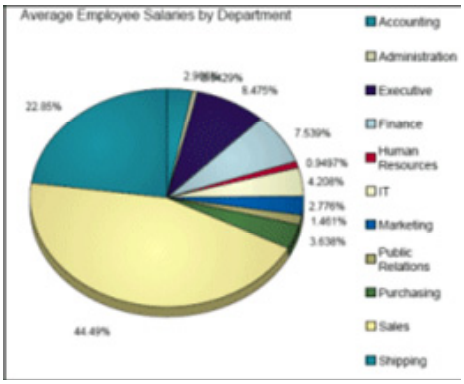
Select a format such as PDF, RTF , HTML, or PowerPoint from the Preview menu to preview the data. Before you preview your work, be sure to save it.

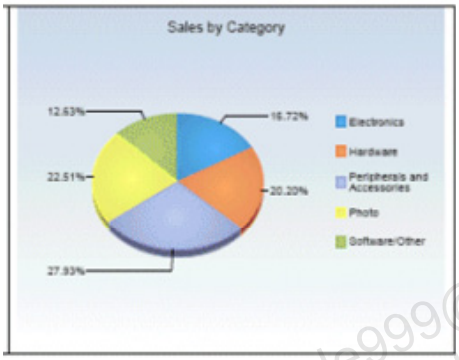
# Practice 5-1: Overview

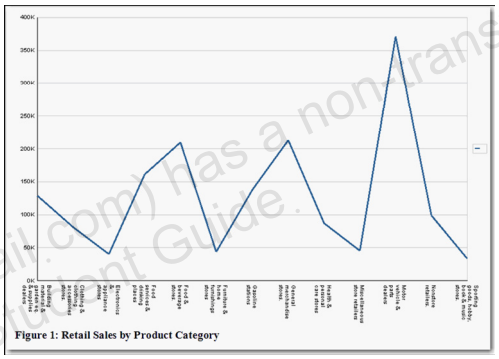
This practice covers creating an RTF template for a sample report.

# BI Publisher Charts



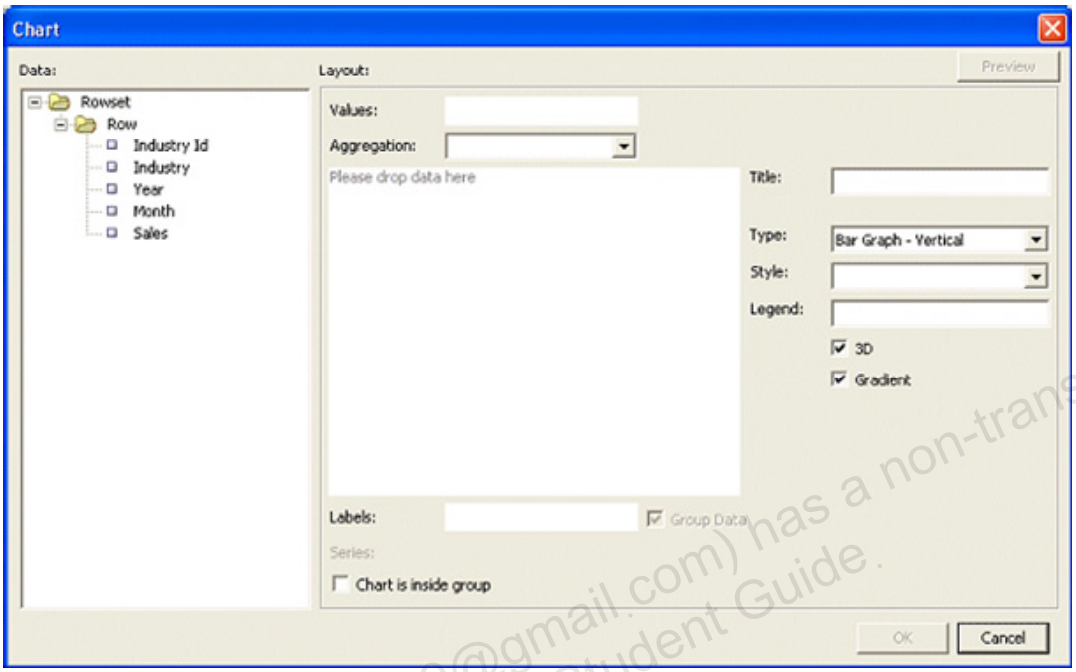



## BI Publisher Charts

BI Publisher's charting feature is based on Oracle Business Intelligence Beans, and it offers all chart types provided by that module. BI Publisher supports almost 70 different graph types and variations, and you can choose one of them for your template. A few sample BI Publisher charts are shown in the slide.

**Note:** Oracle BI Publisher does not support native Microsoft Word charts. You must create charts by using the Oracle BI Publisher Desktop chart wizard (use the Insert > Chart option). Chart types and other Chart options provided in the wizard are discussed in detail on the following pages.

# Creating an RTF Template: Inserting a Chart










## Creating an RTF Template: Inserting a Chart

To insert a chart, you use the following sample files:

- Open the following file: `D:\Program Files\Oracle\BI Publisher Desktop\samples\RTF templates\Sales Report\Retail Sales Start.rtf`
- Load the `RetailSales.xml` file in the D:\Program Files\Oracle\BI Publisher Desktop\samples\RTF templates\Sales Report folder by selecting Load XML Data in the Data menu. (This data contains retail sales data for different industries. You create a chart that shows sales per industry.)

Select Insert > Chart or BI Publisher > Insert > Chart.

Use the Chart dialog box to insert a data-driven chart into a template.

The data source pane shows the structure of the XML data. You notice that you can select only some of the nodes—the data group nodes. Data group nodes (parent nodes) are nodes that have child nodes. They typically do not represent data attributes, but groups of data, such as an invoice, a purchase order, a purchase order line, or a shipment.

Select a data group node in the data source pane that you want to associate with the graph. For example, you may have multiple invoices in a single XML document. If you select the root node containing all invoices, the graph is calculated over all invoices. However, if you select the invoice node, the graph is calculated using the data for a single invoice.

## Creating an RTF Template: Inserting a Chart (continued)

### Chart Options

#### Chart Type

The current version of BI Publisher offers many graph types, and you can choose one of the following options available from the wizard:

- Bar Graph - Vertical
- Bar - Vertical - Percent
- Bar - Vertical - Stacked
- Bar Graph - Horizontal
- Bar - Horizontal - Percent
- Bar - Horizontal - Stacked
- Pie Chart
- Line Graph
- Line-Percent
- Line-Stacked
- Area Graph
- Area-Percent
- Area-Stacked
- Ring Graph
- 3-D Bar Graph
- Radar Graph
- Pareto Graph

#### Values

Drag the data value of interest to the Values field (for example, Revenue).

#### Aggregation

You can choose to aggregate the Values data as a sum, a count, or an average.

#### Labels

Drag the data element for which you want to see the Value charted (for example, Region). Select Group Data to group the label element before rendering it in the chart. For example, if you are charting Revenue by Region, selecting Group Data accumulates the values for Region, so that only one occurrence of each region appears in the chart. If you do not select Group Data, the value for every occurrence of Region in the data is plotted separately.

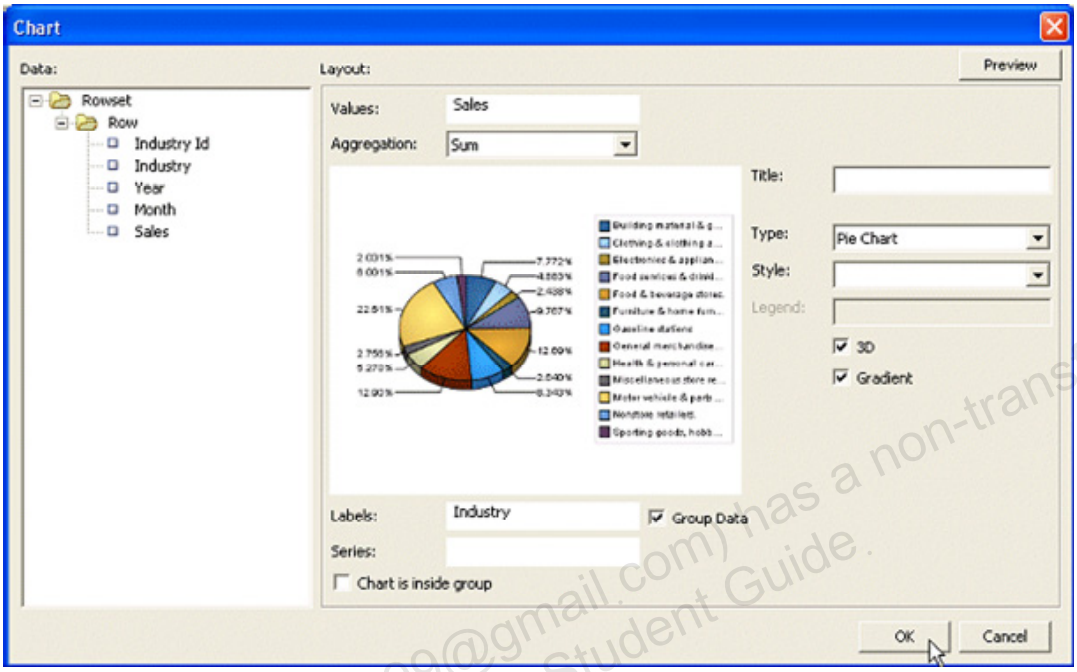
#### Series

Drag the element to display as a series in the chart.

#### Style

Select a color scheme and style for your chart.

# Defining a Chart







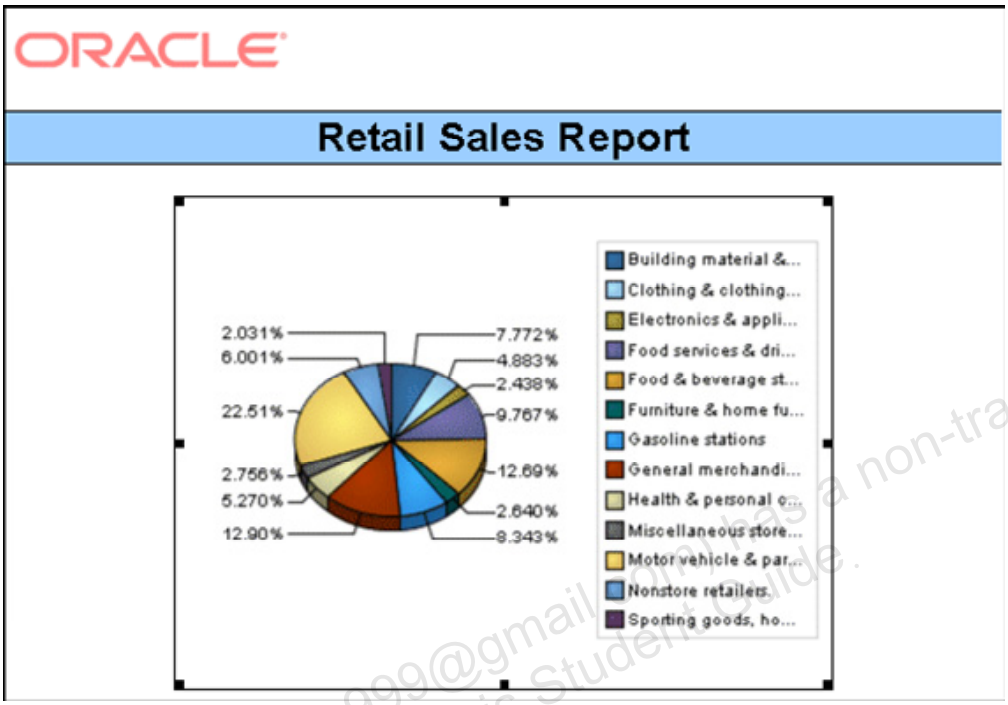



## Defining a Chart

To create the chart as shown in the slide, ensure that Row is selected in the data source pane, and perform the following actions:

- For Type, select **Pie Chart**.
- For Values, select **Sales**.
- For Measure, select **Sum**.
- For Labels, select **Industry**.
- Click **Preview** to preview the graph, and click **OK.**
- You can also select a **Style** to have an appropriate color theme for your graph.

# Previewing the Chart






## Previewing the Chart

Select Preview > PDF/RTF or Oracle BI Publisher > Preview > PDF/RTF.

The slide example shows what the chart looks like if you select the RTF format.

**Note:** Remember to save before you preview.

# Practice 5-2: Overview

This practice covers creating a chart in the RTF template for a sample report

# Designing an RTF Template for a BI Publisher Report

1. In MS Word, create a document and log in to BI Publisher.
2. Open the BI Publisher report (to load XML data).
3. Define an RTF template by using wizards:
   - Add a table.
   - Add a chart.
4. Preview the data in the report by using the template.
5. Upload the template to BI Publisher Enterprise Server, and view data.

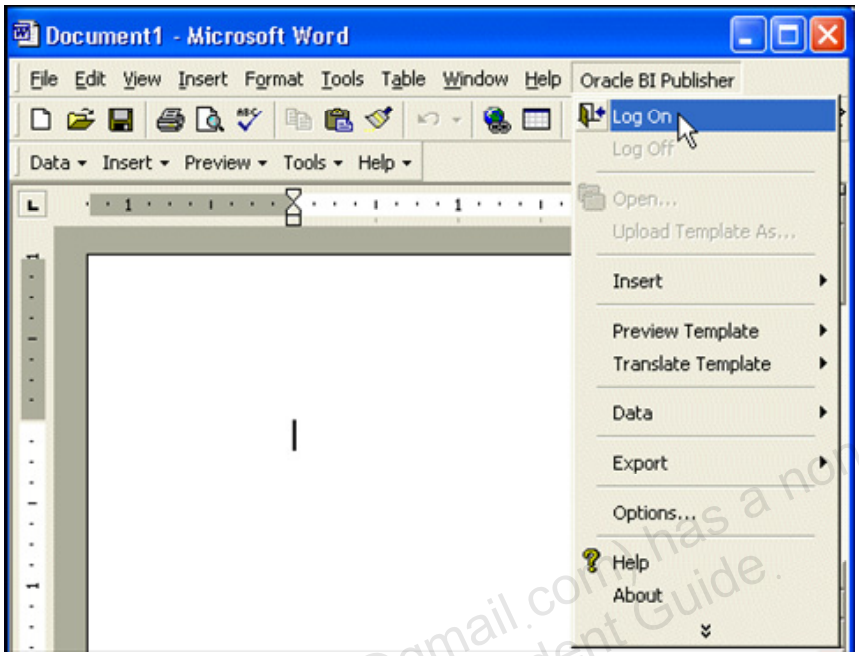




## Designing an RTF Template for a BI Publisher Report

Previously, you created an RTF template for sample data. To create an RTF template for any of the reports created in BI Publisher, you must perform the steps listed in the slide. These steps are discussed in detail on the following pages. To exemplify the steps, the “Based on Oracle DB” report that you have created in the lesson titled “Getting Started with BI Publisher: Creating a Simple Report” is used.

# Step 1: Open MS Word and Log In to BI Publisher







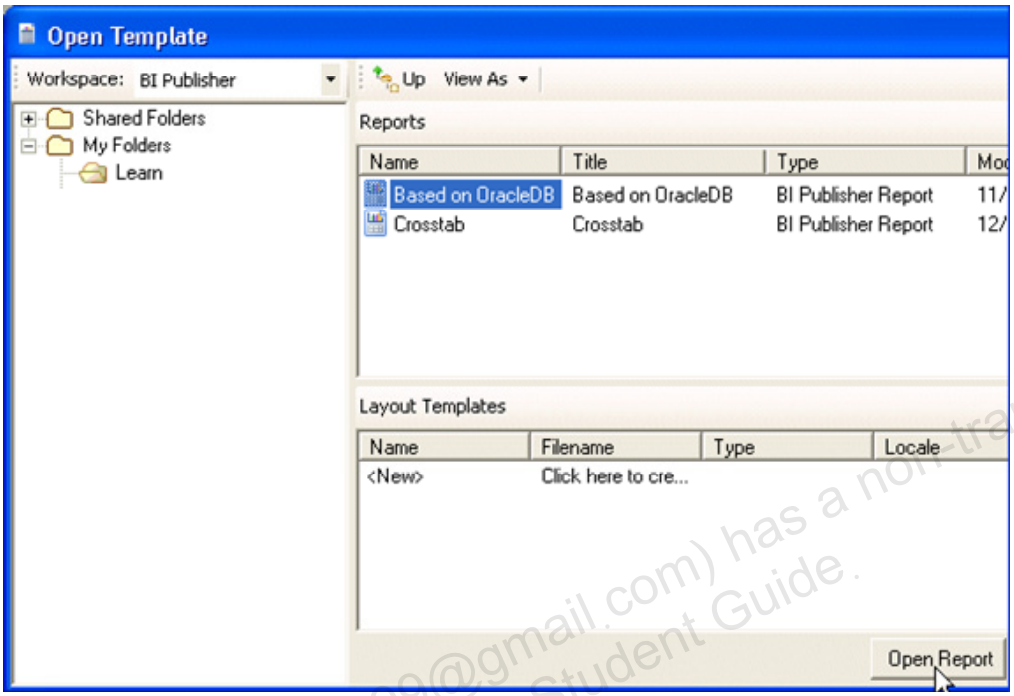



## Step 1: Open MS Word and Log In to BI Publisher

1. Open the MS Word application, and create a document. From the BI Publisher menu, select Log On.
2. On the Login page that appears, enter the username and password (for example, `Administrator/Administrator`), and click Login.
3. The first time you connect to BI Publisher from MS Word, you may be asked for the report server URL. Enter the URL, which is of the format: `http://<Host>:9704/xmlpserver`

When you log in to BI Publisher, the Open Template window is displayed.

# Step 2: Open the BI Publisher Report



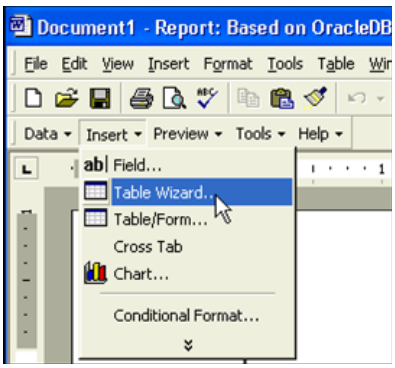


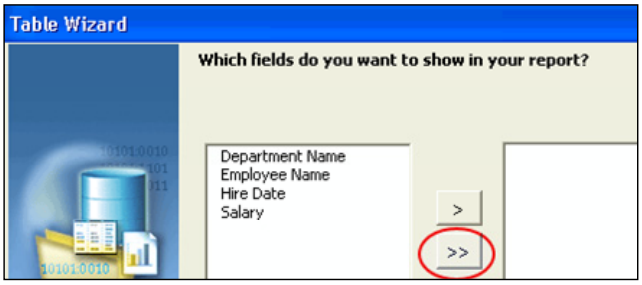

## Step 2: Open the BI Publisher Report

1. In the Open Template window, ensure that BI Publisher is selected from the Workspace drop-down list.
2. Navigate to your working folder (My Folders > Learn) and select the report (in this example, Based on Oracle DB) for which you want to create an RTF template. Click Open Report.

**Note:** When you open the report, you will not see any data in the MS Word document. But this loads the XML data definitions from the report to facilitate the defining of a template. Also, you see that the report name is displayed in the title bar in MS Word beside the document name.

# Step 3: Define the RTF Template: Add a Table





ORACLE







## Step 3: Define the RTF Template: Add a Table

BI Publisher Desktop provides easy-to-use wizards to add tables, forms, charts, and other report elements. Perform the following steps to add a table in the report template:

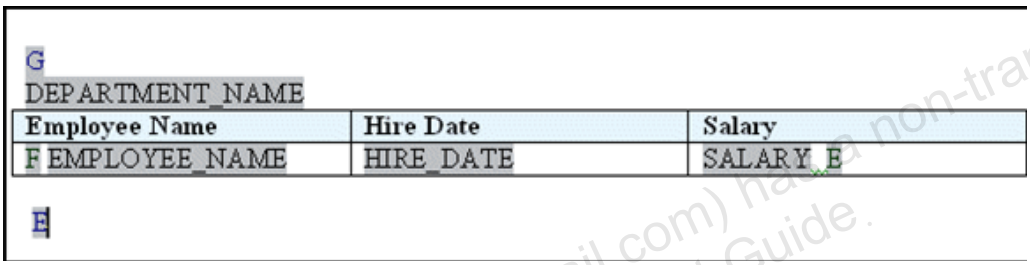
1. Select Insert > Table Wizard from the BI Publisher toolbar to define a table format for the report query. This displays the Table Wizard. Select Table and click Next.
2. Select ROWSET/ROW as the grouping field (this is the default), and click Next.
3. Click Select All (>>) to include all the available columns. Alternatively, use the Select (>) button to select the columns individually. Click Next.
4. Select options in the Group By fields. In the example, select Department Name in the Group By field, and the Group above or Group left options as appropriate. Observe the other options in this step. Click Next.
5. Select options in the Sort By fields and the other Sort options—for example, you may want to sort the report by the descending order of the Salary field or by the ascending order of the Hire Date field. Click Next.
6. Edit the labels of the columns as required, and click Finish.

# Form Fields in RTF Templates

Form fields are Word objects that enable you to refer to other data.

BI Publisher uses form fields to:

- Reference data fields from the report definition
- Embed instructions that control how the data fields are laid out in the table










## Form Fields in RTF Templates

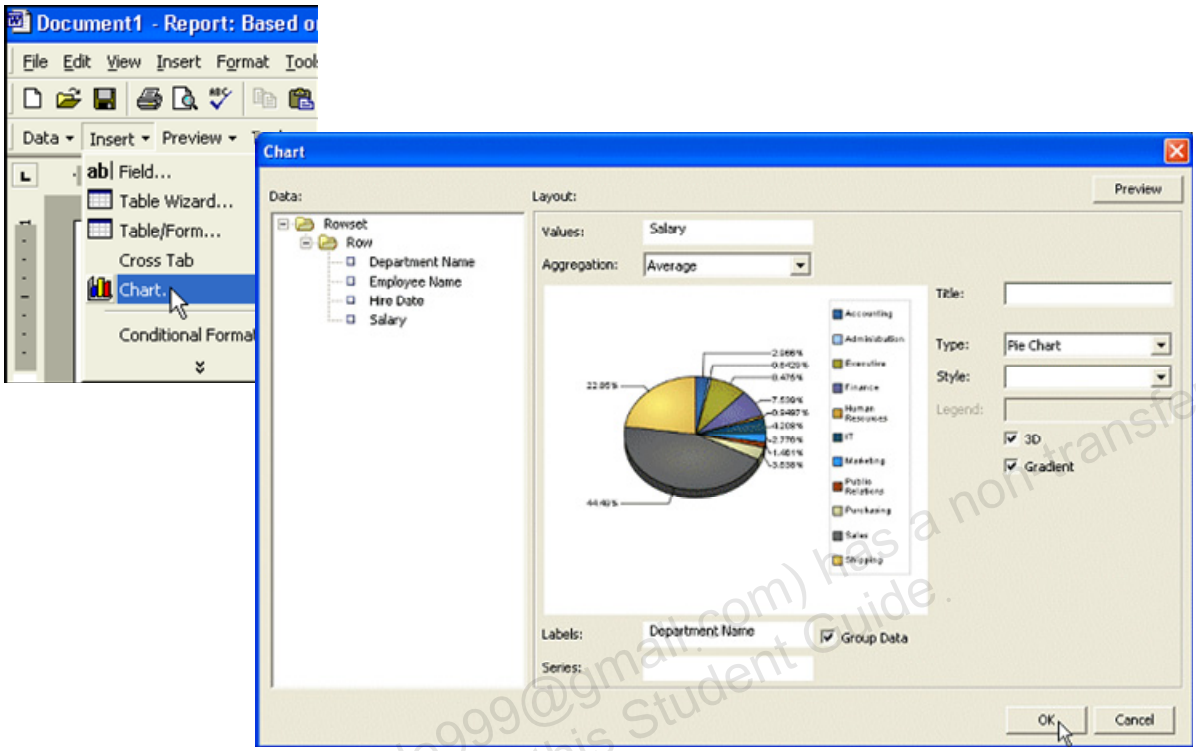
After adding the table, you see that the BI Publisher Desktop Template Builder created a table for you (which looks like the one in the slide). Notice the fields and letters with the gray background. These are called form fields. Form fields are Word objects that enable you to reference other data (for example, a mail merge letter). BI Publisher uses form fields in two ways:

- To reference data fields from the report definition (such as Employee Name, Salary)
- To embed instructions that control how the data fields will be laid out in the table (such as G that stands for "group by," F for "for each," and E for "end").

It is important to treat these form fields carefully and not accidentally delete or move them. Otherwise, the layout of the table in your report will be changed. You can add or modify your own form fields with XSL commands to do more sophisticated things with the table layout.

You can additionally use MS Word's native formatting features in this template, such as changing the background and text colors, adding a title, and so on. This is covered in the lesson titled "Advanced RTF Template Techniques."

# Step 3: Define the RTF Template: Add a Chart






## Step 3: Define the RTF Template: Add a Chart

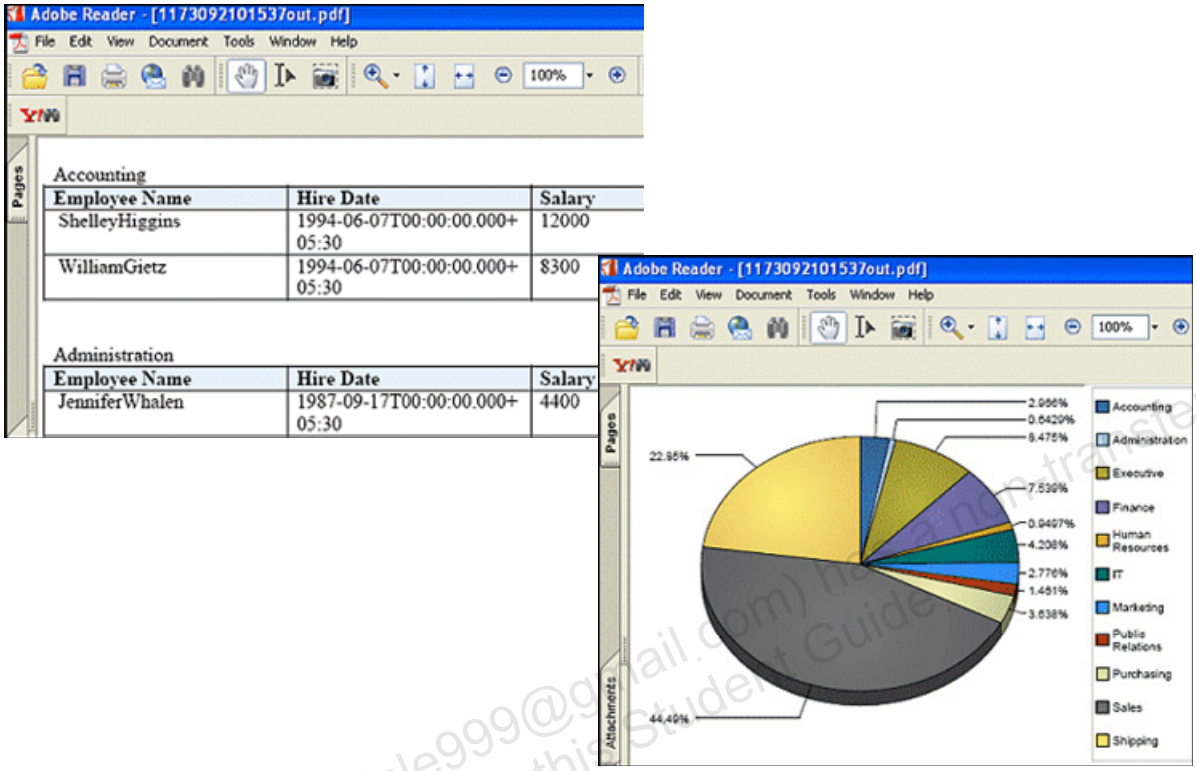
You can also add a chart in the template. (In the example, you can add a chart to display the average employee salaries by department.)

1. To add a chart to the template, select Insert > Chart.
2. In the Chart window that appears, define graph characteristics by performing the following actions:
   - Drag Salary to the Values field and Department Name to the Labels field.
   - Select Average from the Aggregation drop-down list.
   - Select Pie Chart from the Type drop-down list.
   - Click Preview to preview how the graph looks like, and click OK.
3. Save the template as an RTF file.
   - In MS Word, select the File > Save As option.
   - Enter a name for the RTF template file, and click Save.

Do not close the file or log off BI Publisher.

**Note:** You can save the file in any local folder on your system, but you must save the template in the `.rtf` format (not in the `.doc` format).

# Step 4: Preview the Data by Using the Template



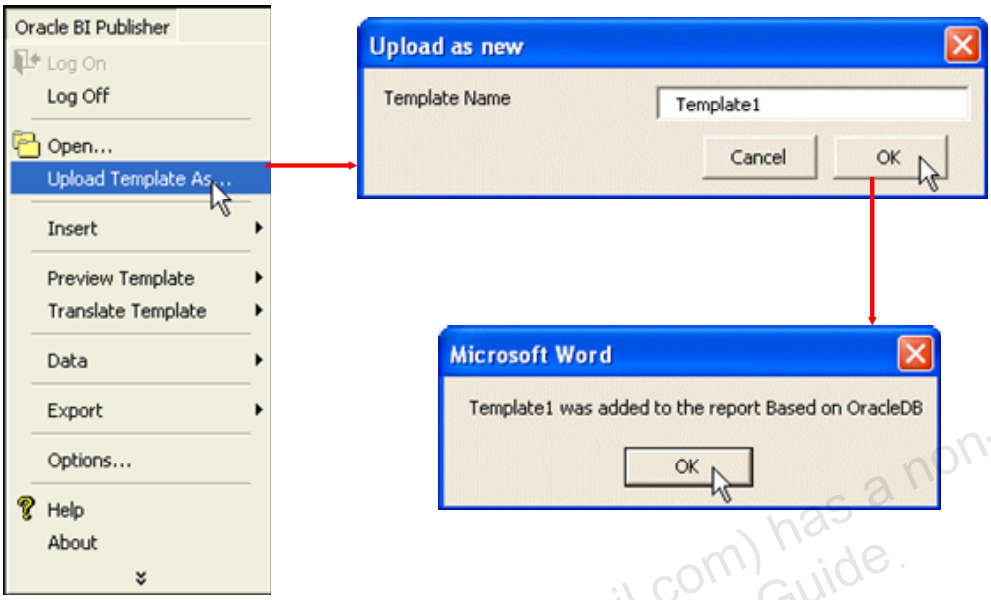



## Step 4: Preview the Data by Using the Template

Before you publish the report, you can see how data in the report is displayed by using the template. Select Preview > PDF to preview data by using the template you created.

The data and graph are shown in the slide.

# Step 5: Upload the Template and View Data






## Step 5: Upload the Template and View Data

1. In MS Word, select Oracle BI Publisher > Upload Template As.
   **Note:** If you have not saved the template in RTF format, it may prompt you to save the template in RTF format before publishing the template.
2. The "Upload as new" dialog box appears. Enter a name for the template, and click OK.
3. After the template is uploaded, a confirmation message indicating that the template is added to the BI Publisher report is displayed. Click OK again.
4. You can now log in to BI Publisher Enterprise and view the report by using the template uploaded.

# Practice 5-5: Overview

This practice covers creating and publishing an RTF template for the BI Publisher report that you created in Practice 4.

# Methods for Creating RTF Templates

Although there are virtually unlimited ways to create RTF templates for Oracle BI Publisher, only two methods are supported:

- Basic RTF method
- Form field method





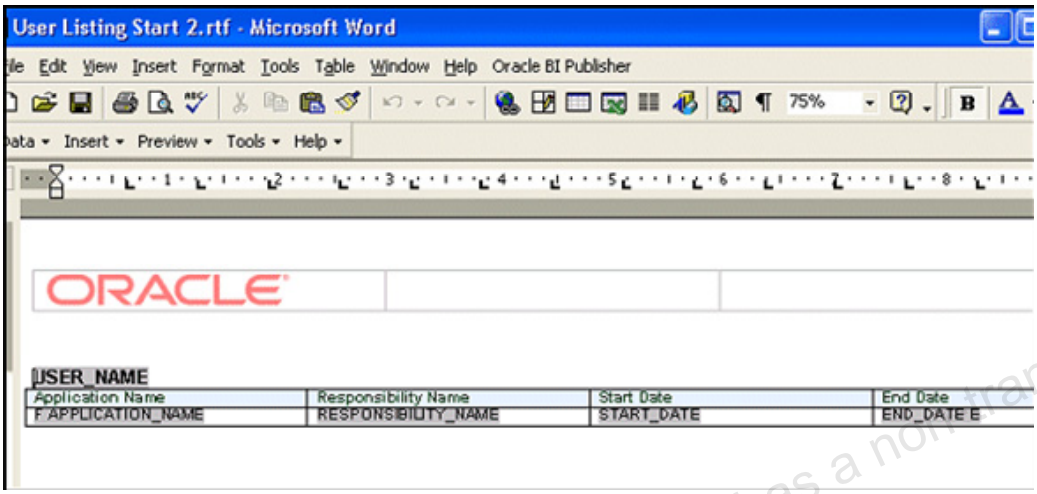


## Methods of Creating RTF Templates

So far you have seen how you can build up on an existing template. BI Publisher supports mainly two methods for creating RTF templates:

- **Basic RTF method:** With any word-processing application that supports RTF version 1.6 (or later), design a template by using Oracle BI Publisher's simplified syntax.
- **Form Field method:** Using Microsoft Word's form field feature, place the syntax in hidden form fields rather than directly into your design. Oracle BI Publisher supports this using Microsoft Word 2000 (or later) on a system that runs Microsoft Windows 2000 (or later). The form field method relies upon the Oracle BI Publisher Desktop plug-in that works only with Microsoft Word 2000 and later.

# Basic Method: Example










## Basic Method: Example

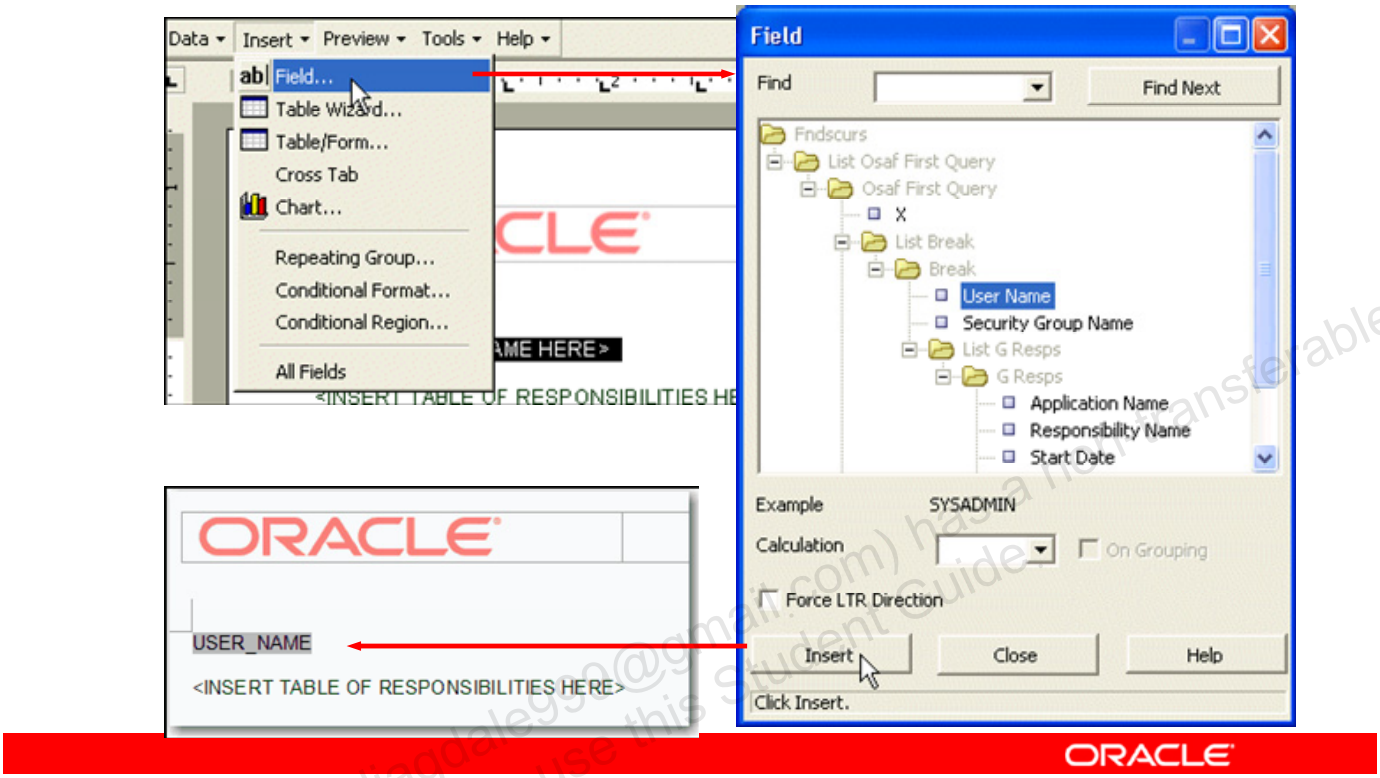
In your lab environment, the `User Listing Start 2.rtf` file can be found at the following directory path: `D:\Labs\ BIP_10.1.3.3._Labs\Templates\User Listing\`

For this class, you use Microsoft Word to create and edit your RTF files. But any program capable of creating and editing RTF files can be used for the basic method.

Start with the `User Listing Start 2.rtf` file. Edit the RTF template by using the basic method to place the fields:

1. In Microsoft Word, open the `User Listing Start 2.rtf` file.
2. In the template, replace all text in < > (angle brackets) with the appropriate Oracle BI Publisher tags.
3. Enter `<?USER_NAME?>` in the User Name field.
4. Enter `<?for-each:G_RESPS?>` and `<?APPLICATION_NAME?>` in the Application Name column of the table.
5. Enter `<?RESPONSIBILITY_NAME?>` and `<?START_DATE?>` in the corresponding columns.
6. Enter `<?END_DATE?>` and `<?end for-each?>` in the End Date column of the table.
7. Save the new template as `User Listing End 2.rtf`.
8. Exit Microsoft Word.

# Form Field Method: Example










## Form Field Method: Example

In your lab environment, the `User Listing Start.rtf` file can be found at the following directory path: `D:\Labs\ BIP_10.1.3.3._Labs\Templates\User Listing\`

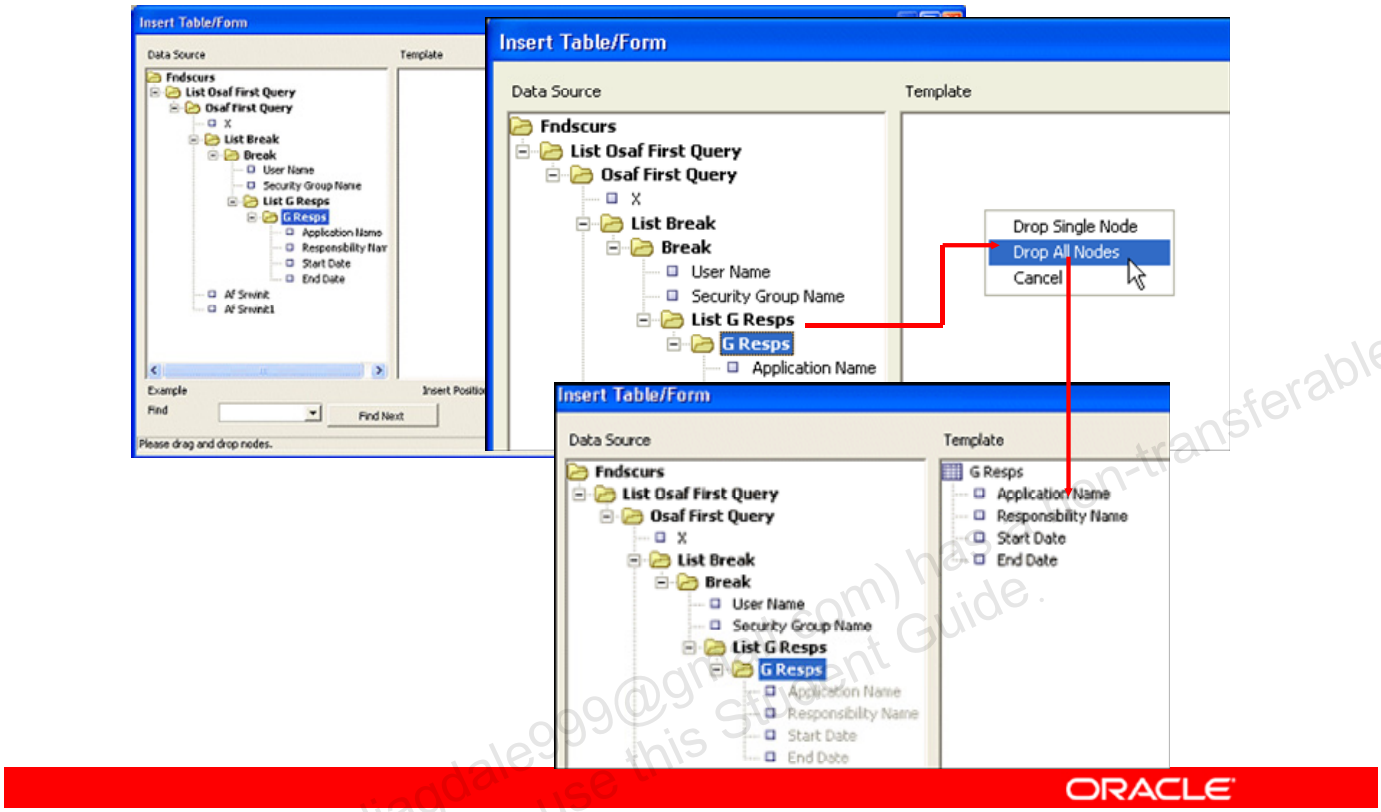
For this class, you use Microsoft Word to create and edit RTF files. The form field method relies on the Oracle BI Publisher Desktop plug-in that works only in Microsoft Word 2000 and later.

Start with the `User Listing Start.rtf` file. Edit the RTF template by using the form field method to place the fields:

1. In Microsoft Word, open the `User Listing Start .rtf` file.
2. Load the XML data file `FNDSCURX.xml` by selecting Data > Load XML Data.
3. In the template, replace all the text in < > (angle brackets) with appropriate Oracle BI Publisher tags.
4. Now insert fields in the templates. Select the `<INSERT USER NAME HERE>` placeholder in the template file, and from the Oracle BI Publisher Desktop menu, select Insert > Field. When the Field selector opens, select USER_NAME from the list, and then click Insert.

   **Note:** You can see that the USER_NAME field is inserted in the RTF template. Notice that the USER_NAME field is gray, indicating that it is a form field.

# Form Field Method: Creating a Data Table



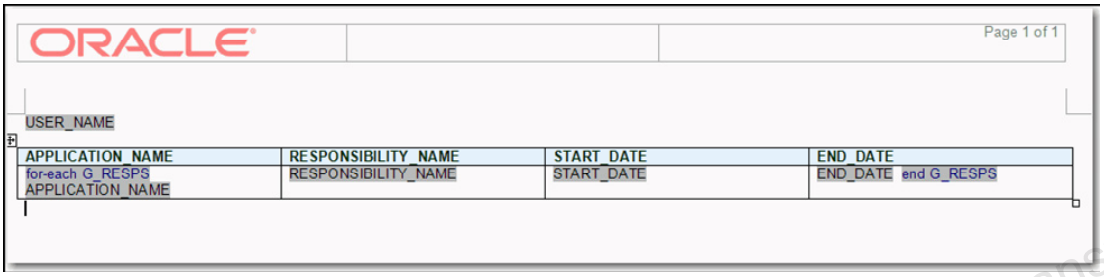



## Form Field Method: Creating a Data Table

5. Create a data table for the rest of the data. Unlike the basic method, you are not going to start with a table created for you. You are going to let Oracle BI Publisher Desktop create one for you.
6. Choose Insert > Table/Form. (The Insert Table/Form dialog box appears.)
7. Click G_RESPS in the Data Source pane, and drag this group to the Template pane.
8. You are prompted to select Drop Single Node, Drop All Nodes, or Cancel. Select Drop All Nodes to grab the four fields beneath G_RESPS.
9. Click OK in the Insert Table/Form window to save your work and then return to the template. (The images in the slide show what the Template pane looks like when you drop all the nodes.)
10. Save your template as `User Listing End.rtf`.

# Completed Template

Your completed template should look similar to the following:






ORACLE



## Completed Template

When you return to your template, note that the wizard created the table, the fields in the table, and the repetitive section defined with the `<for-each>` and `<end>` tags.

# Previewing the Report

| SYSADMIN | | | |
|---|---|---|---|
| **Application Name** | **Responsibility Name** | **Start Date** | **End Date** |
| Activity Based Management | ABM Manager | 11-FEB-00 | |
| Activity Based Management | ABM Supervisor | 19-APR-00 | |
| Alert | Alert Manager, Vision Enterprises | 01-JAN-51 | |
| Application Implementation | Oracle iSetup | 26-NOV-02 | |
| Application Object Library | Application Developer | 01-JAN-51 | |
| Application Object Library | Functional Administrator | 28-JAN-03 | |
| Application Object Library | Functional Developer | 28-JAN-03 | |
| Application Object Library | User Management | 10-MAR-04 | |
| Application Object Library | Workflow Administrator Web Applications | 09-JUN-99 | |
| Application Object Library | Workflow User Web Applications | 09-JUN-99 | |
| Applications BIS | Business Intelligence System, Vision Operations (USA) | 01-JAN-51 | |
| Applications BIS | Business Views Setup | 01-JAN-51 | |
| Applications BIS | Custom AOL Workbooks | 11-NOV-04 | |
| Applications BIS | Daily Business Intelligence Administrator | 22-MAY-02 | |
| Applications BIS | Daily Business Intelligence Designer | 23-JUL-04 | |
| Balanced Scorecard | Performance Management Designer | 06-DEC-01 | |
| Balanced Scorecard | Performance Management Security Administrator | 30-JUN-03 | |
| Balanced Scorecard | Performance Management User | 30-JUN-03 | |
| CRM Foundation | CRM Administrator, Vision Enterprises | 15-FEB-51 | |
| CRM Foundation | CRM ETF ADMINISTRATION | 11-JUN-01 | |
| CRM Foundation | CRM HTML Administration | 19-APR-00 | |
| CRM Foundation | CRM HTML Developer | 11-JUN-01 | |
| CRM Foundation | Interaction History JSP Admin | 03-DEC-01 | |
| Common Modules-AK | Ak Html Forms | 11-SEP-00 | |
| Common Modules-AK | Application Developer Common Modules | 01-JAN-51 | |
| Human Resources | HRMS Manager, Vision University | 01-JAN-97 | |
| Install Base | Install Base Admininistrator | 25-APR-01 | |
| Install Base | Install Base User | 15-MAY-01 | |



## Previewing the Report

You can preview your results in Microsoft Word. Select Preview > PDF to view the report in PDF. (You can also select any of the other supported formats such as HTML, RTF, EXCEL, PowerPoint, and so on.)

If you have done everything correctly, Adobe Acrobat Reader should open with the output of your report. If not, you have done something in error and you must return to Microsoft Word to edit your file until it produces the correct output.

# Practices 5-3, 5-4: Overview

These practices cover creating an RTF template for a report by using the following methods:

- Basic (Practice 5-3)
- Form field (Practice 5-4)

# Summary

In this lesson, you should have learned how to:

- Describe the functions and features of BI Publisher Desktop
- Describe how to install BI Publisher Desktop
- Create RTF templates for a sample report
- Create and publish RTF templates for BI Publisher reports
- Create RTF templates by using the following methods:
  - Basic
  - Form field
- Insert tables, forms, and charts in RTF templates
- Preview the results

# Practice 5: Overview

This practice covers the following topics:

- Creating simple RTF templates by using sample reports or files
- Creating RTF templates by using basic and form field methods
- Creating and publishing RTF templates for BI Publisher reports
- Adding fields, table, and charts to RTF templates
- Modifying RTF templates
- Previewing results

# 6

# Advanced RTF Template Techniques

# Objectives

After completing this lesson, you should be able to do the following:

- Use native MS Word features to format RTF templates
- Use other advanced features

# Know Your Data

Oracle BI Publisher does not provide an automated (intelligent) mapping mechanism. There are two useful methods for exploring and understanding data:

- Looking at raw XML data
- Looking at the data structure in Oracle BI Publisher Desktop







## Know Your Data

In the lesson titled “Creating Simple RTF Templates,” you explicitly included certain fields or typed certain tags, such as <?USER_NAME?>. But how do you know what to enter or what to include using the drag-and-drop operation? BI Publisher does not provide an automated mapping mechanism; you must understand the data you are using and then correctly map it so that it appears properly in your report.

Your template content and layout must correspond to the content and hierarchy of the input Extensible Markup Language (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. Specifically, to map data fields, you define *placeholders*. To designate repeating elements, you define *groups*.

# Looking at Raw XML Data

This snippet of XML data is a sample of what you may be provided:

```
<?xml version="1.0"?>
<!-- Generated by Oracle Reports version 6.0.8.25.0 -->
<FNDSCURS>
  <LIST_OSAF_FIRST_QUERY>
    <OSAF_FIRST_QUERY>
      <X>X</X>
      <LIST_BREAK>
        <BREAK>
          <USER_NAME>SYSADMIN</USER_NAME>
          <SECURITY_GROUP_NAME>Standard</SECURITY_GROUP_NAME>
          <LIST_G_RESPS>
            <G_RESPS>
              <APPLICATION_NAME>Activity Based Management</APPLICATION_NAME>
              <RESPONSIBILITY_NAME>ABM Manager</RESPONSIBILITY_NAME>
              <START_DATE>11-FEB-00</START_DATE>
              <END_DATE></END_DATE>
            </G_RESPS>
```







## Looking at Raw XML Data

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 XML elements. The placeholder maps the template report field to the XML element. At run time, the placeholder is replaced by the value of the element of the same name in the XML data file.

The slide shows a good example of data coming from an outside source. In this case, the data was produced by Oracle Reports running from within an Oracle 11*i* E-Business Suite database and querying the user data from that system. But it is not uncommon for XML coming from other data systems to produce unnecessary XML tags before the first tag of interest. Most raw XML data can be difficult to read (except the simplest data). Because it is difficult, you may make errors in interpreting the data.

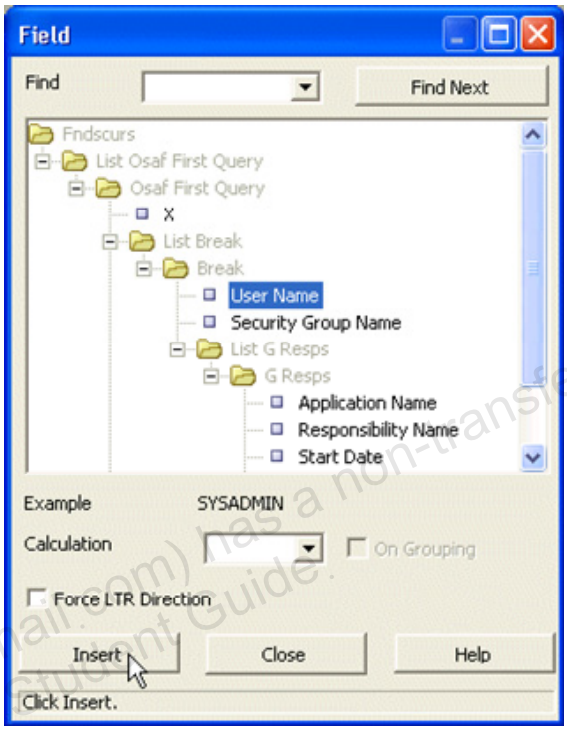
In this data, the first tag of interest is the <USER_NAME>SYSADMIN</USER_NAME> tag.

Notice the group data, <G_RESPS>, beneath the <USER_NAME> tag.

**Note:** Only a portion of the file is shown in the slide, so all the closing tags are not present.

# Looking at the Data Structure in Oracle BI Publisher Desktop

The Field dialog box (accessed by selecting Insert > Field) is less error-prone. You can see tabular structures more easily.










## Looking at the Data Structure in Oracle BI Publisher Desktop

Select Insert > Field to see the data structure in BI Publisher.

In the Field dialog box, you can select data elements defined in the data source (that is, the XML file) and insert them into the template.

The dialog box shows the structure of the data source in a tree view. As you have already learned, you select a field that represents a single data field (a leaf node of the tree) and click Insert to place a text form field with hidden BI Publisher commands (in the Help text of the form field) at the cursor position in the template.

**Note:** You can also select Insert >Table/Form to see the data structure.

# Underlying Tags

In the basic method:

XML data = <USER_NAME>

Placeholder = <?USER_NAME?>

But how does the form field method create tags?



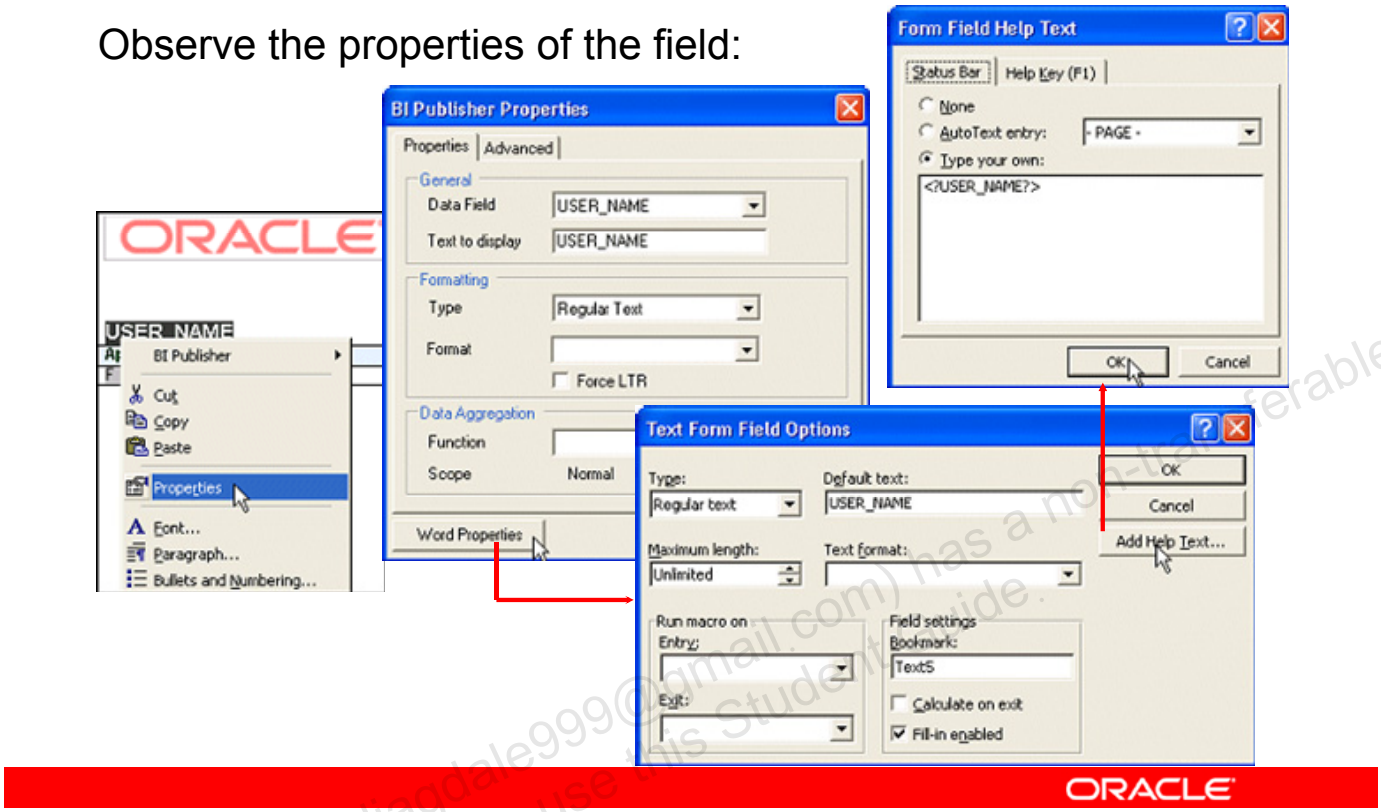




## Underlying Tags

In the basic method, it is easy to see the relationship between the XML data and the tag as indicated in the slide. The form field tags are discussed on the next page.

# Form Field Method Tags






## Form Field Method Tags

You can view Form Field tags by selecting Properties of a field. Right-click a field and select Properties to open the BI Publisher Properties. Observe the general and advanced properties. Also, click Word Properties to see Text Form Field Options, and click Add Help Text to see the Form Field Help Text. Note that the placeholder, even with the form field method, is still <?USER_NAME?>, which is what it would be if you used the basic method.

# Additional Tag Space

There are two places to put tags:

- Status Bar (first)
- Help Key (F1) (second)







## Additional Tag Space

You can add tags at two places: Status Bar and Help Key (F1). However, the form field method has a limitation. The Status Bar part of Additional Help Text is limited to 138 characters. The Help Key (F1) part is limited to 255 characters. These two areas are simply concatenated with Status Bar appearing first, followed by Help Key (F1). So, in total, you are limited to 393 characters. If you have placeholders that stretch capabilities, or if you use advanced functions that go beyond 393 characters, you must use the basic method.

**Pre-markup Layout:**
Oracle BI Publisher converts all formatting that you apply in the word processing program to XSL-FO, but some features of your word processor may not be mappable to XSL-FO.

# RTF Template: Design Considerations

- Premarkup layout
- Supported MS Word–native formatting features
- Adding markup
- Images and charts
- Template features
- Conditional formatting
- Page-level calculations
- Data handling
- Variables, parameters, and properties
- Advanced report layouts





## RTF Template: Design Considerations

Consider the points in the slide for creating an efficient design for RTF templates. BI Publisher supports the use of all these features. Most of these features are also supported in BI Publisher Desktop. More details about advanced template techniques are presented on the following pages.

**Note:** For more information about the supported features for designing templates, see the *Oracle BI Publisher User's Guide*.

# Supported MS Word Native Formatting Features

- Alignment
- Fonts, background, text colors
- Table formats
- Inserting clip arts and images
- Inserting headers and footers
- Date fields
- Including watermarks
- Page breaks, and so on







## Supported MS Word Native Formatting Features

BI Publisher supports many native formatting features of MS Word when you use BI Publisher Desktop to create and edit templates. A few of these features are listed here:

**General features:**

- Large blocks of text
- Fonts, background, and text colors
- Page breaks and page numbering
- Alignment (supported for text, graphics, objects, and tables)
- Bidirectional languages handled automatically using a word processing application's left and right alignment controls

**Table formatting:**

- Nested tables, autoformatting of tables, and repeating table headers
- Cell alignment, cell patterns, and colors
- Row and column spanning
- Rows prevented from breaking across pages
- Fixed-width columns
- Truncated fields

**Date fields:** They correspond to the publishing date, and not the request run date.

**Multicolumn page support:** This enables you to use the multiple column function from Microsoft Word to publish your output.

# Adding Markup

Additional markup features:

- Placeholders:
  - `<?XML element tag name?>`
- XML data value (example):
  - `<?USER_NAME?>`
- Groups (example):
  - `<?for-each:USER_NAME?>`
  - `<?end for-each?>`

Advanced markup features:

- Headers and footers:
  - Use native RTF headers and footers.
- Subtemplates







## Adding Markup

Additionally, you can add the markup (for placeholders, groups, and so on) in the templates. You can easily mix manual markup and the markup done in Oracle BI Publisher Desktop. In fact, Oracle BI Publisher Desktop simply uses the hidden form field method available in Microsoft Word 2000 (and later).

### 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.

### 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 XML elements. The placeholder maps the template report field to the XML element. At run time, the placeholder is replaced by the value of the element of the same name in the XML data file.

## Adding Markup (continued)

### Defining Groups

By defining a group, you are notifying Oracle BI Publisher that *for each* occurrence of an element (parent), you want the included fields (children) displayed. At run time, Oracle BI Publisher loops through the occurrences of the element and display the fields each time. To designate a group of repeating fields, insert the grouping tags around the elements to repeat.

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

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

### Headers and Footers

Oracle BI Publisher supports the use of the native RTF header and footer feature. To create a header or footer, use the word processing application's header and footer insertion tools. Alternatively, you can define an internal template in the body of your template, and then call this subtemplate from the header or footer region.

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

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

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

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 header-and-footer template alternative.

If you require many or complex objects in the header or footer of your report, these regions can become difficult to read or understand. Alternatively, you can create header-and-footer templates in the body of your RTF template document that can then be called from the header or footer region.

For more information, refer to the *Oracle BI Publisher User's Guide*.

## Adding Markup (continued)

### Subtemplates

To create a template within your template, wrap the contents of the internal template with the following tags:

- `<?template:internaltemplate name?>`
- `<?end template?>`

  In this code, `internaltemplate name` is the name that you assign to the header/footer template.

Then in the header/footer region, enter the following syntax to call this template:

- `<?call:internaltemplate name?>`

At run time, the contents of the internal template are rendered at the position of the call.

### Example

Assume that you have the following XML data:

```
<REPORT>
    <COMPANY_NAME>Oracle</COMPANY_NAME>
    <REPORT_NAME> Accounts Payables - Invoice Listing</REPORT_NAME>
    <REPORT_DATE>1st January 2005</REPORT_DATE>
    <LEGAL_TEXT>Private and Confidential</LEGAL_TEXT>
...
/<REPORT>
```

You want `COMPANY_NAME`, `REPORT_NAME`, and `REPORT_DATE` to be displayed in a table in the header of the report. In the header of your template, enter the following:

```
<?call:OraHeader/>
```

At run time, the OraHeader template is rendered in the header of each page of the document.

# Images and Charts

Oracle BI Publisher supports several methods for including images and charts in an Oracle BI Publisher report:

- Direct insertion
- Images in the form of clip arts
- The URL reference
- The OA_MEDIA directory reference
- Leveraging Oracle Business Intelligence Beans (BI Beans) to add charts and graphs
- Image retrieved from binary large object (BLOB) data







## Images and Charts

BI Publisher supports several methods for including images or charts.

**Direct Insertion**

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

**Clip Arts**

You can insert images by using the MS Word Clip Art libraries in your report templates.

**URL Reference**

- Insert a dummy image in your template. For layout purposes, it is best if the dummy image is the same size, in pixel height and width, as your actual image.
- In the Format Picture dialog box (right-click the image to open it), click the Web tab. Enter the following syntax in the Alternative Text region:
  `url:{'http://image.location.com/image-name'}`

**OA_Media Directory Reference**

The OA_MEDIA method refers to only Oracle E-Business Suite installations. Otherwise, it is the same as the URL reference with Alternative Text changing to the following:
url:{'${OA_MEDIA}/image-name'}

## Images and Charts (continued)

### 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 run time. BI Publisher supports all the graph types and component attributes available from the BI Beans graph document type definition (DTD). Most of these are provided through the Chart Wizard in BI Publisher Desktop. Alternatively, you can also use XSL to define the charts in templates. At run time, Oracle BI Publisher calls the appropriate BI Bean to render the image that is inserted as a graph or chart.

The BI Beans graph DTD is fully documented in the following technical note available in the Oracle Technology Network:

http://www.oracle.com/technology/products/reports/htdocs/getstart/whitepapers/graphdtd/graph_dtd_technote_2.html

There is also information published on the BI Publisher blog:

http://blogs.oracle.com/xmlpublisher/

### Rendering an Image Retrieved from BLOB Data

If your data source is an Oracle BI Publisher Data Template, 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 run time:

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

Here, `image/jpg` is the Multipurpose Internet Mail Extensions (MIME) type of the image (other options may be `image/gif` and `image/png`) and `IMAGE_ELEMENT` is the element name of the BLOB in your XML data.

You can specify height and width attributes for the image to set its size in the published report. Oracle BI Publisher scales the image to fit the box size that you define. For example, to set the size of the preceding example to three inches by four inches, enter the following:

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

Specify the size in pixels:

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

You can also specify the size in centimeters:

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

You can also specify the size as a percentage of the original dimensions:

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

# Adding a Chart

- Using the Chat Wizard in BI Publisher Desktop:
  - Select the appropriate options in the wizard to select the graph type, series, and color themes.
- Using Extensible Stylesheet Language (XSL) commands:
  - Insert a dummy image in your template to define the size and position of the chart.
  - Add a definition for the chart, to the alternative text box of the dummy image, by using XSL commands.







## Adding a Chart

BI Publisher supports almost 70 different graph types and variations, and you can choose one of them for your template. Chart types and other chart options are discussed in detail in the lesson titled "Creating Simple RTF Templates." You can add charts in report templates by using the Chart Wizard in BI Publisher Desktop (already discussed in the lesson titled "Creating Simple RTF Templates") or XSL commands.

# Support for Drawings and Shapes

The following AutoShape categories are supported:

- Lines: Straight, arrowed, connectors, curve, free-form, and scribble
- Connectors: Only straight connectors
- Basic Shapes: All shapes
- Block Arrows: All arrows
- Flowchart: All objects
- Stars and banners: All objects
- Callouts: All callouts except “Line” callouts





**Support for Drawings and Shapes**

BI Publisher also supports the use of different drawings and shapes. These are mentioned in the slide.

# Other Graphic Features

- Freehand drawing: Supported
- Hyperlinks: Supported
- Layering: Supported
- 3-D effects: Not currently supported
- Microsoft Equation: Supported
- Organization Chart: Supported
- WordArt: Mostly supported

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





ORACLE



## Other Graphic Features

- **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.
- **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.
- **3-D Effects:** BI Publisher does not currently support the 3D option for shapes. (But the 3D option is supported in the Chart Wizard of BI Publisher Desktop.)
- **Microsoft Equation:** Use the equation editor to generate equations in your output.
- **Organization Chart:** Use the organization chart functionality in your templates and the chart will be rendered in the output.
- **WordArt:** You can use Microsoft Word's WordArt functionality in your templates.

**Note:** Some Microsoft WordArt uses bitmap operations that currently cannot be converted to Scalable Vector Graphics (SVG). To use the unsupported WordArt in your template, take a screenshot of the WordArt and save it as an image (`.gif`, `.jpeg`, or `.png`). Replace the WordArt with the image.

# Data-Driven Shape Support

The following manipulations of shapes are supported:

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







## Data-Driven Shape Support

In addition to supporting static shapes and features in your templates, Oracle BI Publisher supports the manipulation of shapes based on incoming data or parameters.

Enter manipulation commands for a shape on the Web tab of the shape's properties dialog box (right-click and select Format AutoShape > Web > Alternative Text).

### Replicating a Shape

You can replicate a shape based on incoming XML data in the same way that 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, use this syntax to replicate a shape down the page:

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

In this example, `for-each@shape` opens the `for-each` loop for the shape context, and `SHAPE_GROUP` is the name of the repeating element from the XML file. For each occurrence of the `SHAPE_GROUP` element, a new shape is 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, and so on); 1 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. Subsequently, each offset would be another 100 pixels down.

## Data-Driven Shape Support (continued)

### Adding 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 properties dialog box, enter the following syntax:

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

Here, `SHAPETEXT` is the element name in the XML data. At run time, the text is inserted into the shape.

### Adding 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 properties dialog box, enter:

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

Here, `SHAPETEXT` is the element from the XML data. At run time, the value of the `SHAPETEXT` element is 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 pixels along the x-axis, enter the following commands in the properties dialog box of the shape:

```
<?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:

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

Here, `ANGLE` is the number of degrees to rotate the shape. If the angle is positive, the rotation is clockwise; if it is 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.

### Skewing a Shape

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

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

Here, `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. The default is left/top.

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

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

## Data-Driven Shape Support (continued)

### Changing the Size of a Shape

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

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

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

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

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

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

# Background and Watermark Support

Oracle BI Publisher supports:

- Adding a color as background in Microsoft Word
- Adding text or image watermark (in Microsoft Word 2002 or later)

**Note:** This feature is supported for PDF output only.







### Background and Watermark Support

You can specify a single, graduated color or an image background for your template to be displayed in the PDF output. However, this feature is supported for PDF output only.

To add a background by using Microsoft Word 2000, select Format > Background.

On the Background menu, you can:

- Select a single-color background from the color palette
- Select Fill Effects to open the Fill Effects dialog box

In Microsoft Word 2002 or later, you can also add a text or image watermark by selecting Format > Background > Printed Watermark.

# Using More Advanced Template Features

- Page breaks
- Initial page numbers
- Hyperlinks
- Table of contents
- Bookmarks in PDF output
- Check boxes
- Drop-down lists







## Using More Advanced Template Features

Most of the advanced features discussed in this lesson are supported in BI Publisher wizards. The examples also show the tags or aliases for using these features with the templates.

### Page Breaks

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

You can also insert a form field immediately following the form field of the element after which you want the page break to occur. In the Help text of this form field, enter the following:

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

### Page-Level Calculations

You can create page-level calculations in BI Publisher Desktop, and mention the aggregation type that you want to specify for the field; for example, sum of sales, average employee salary. You can use markup to specify page-level calculations. These are discussed later in the lesson.

## Using More Advanced Template Features (continued)

### Initial Page Numbers

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. With Oracle BI Publisher, you can 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:** Suppose that your XML data contains an element to carry the initial page number, as in this example:

```
<REPORT>
     <PAGESTART>200<\PAGESTART>
     ....
</REPORT>
```

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:** Suppose that you set the 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" in the *Oracle BI Publisher User's Guide*.

### Hyperlinks

You can add fixed or dynamic hyperlinks to your template:

- To insert static hyperlinks, use your word processing application's "insert hyperlink" feature.
- If your template includes a data element that contains a hyperlink or part of one, you can create dynamic hyperlinks at run time. In the "Type the file or Web page name" field of the Insert Hyperlink dialog box, enter the following:

  ```
  {URL_LINK}
  where URL_LINK is the incoming data element name.
  ```

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

  ```
  http://www.oracle.com?product={PRODUCT_NAME}
  where PRODUCT_NAME is the incoming data element name.
  ```

- In both these cases, the dynamic URL is constructed at run time. (This topic is discussed in more detail in the lesson titled "Creating Parameters, List of Values (LOVs), and Hyperlinks.")

### Table of Contents

Oracle BI Publisher supports the RTF specification's feature for generating a table of contents. Follow your word processing application's procedures for inserting a table of contents. Oracle 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, perform the following steps:

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 by using your word processing application's table of contents feature.
3. At run time, the TOC placeholders and heading text are 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.

- To create links for a static table of contents, enter:

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

- To create links for a dynamic table of contents, enter:

```
<?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 selected (Checked) or deselected (Not Checked) based on a value from the incoming data. To define a check box in your template, perform the following steps:

1. Position the cursor in your template where you want the check box to appear, and select the Check Box Form Field on the Forms toolbar.
2. Right-click the field to open the Check Box Form Field Options dialog box.
3. Specify the default value as either Checked or Not Checked.
4. In the Form Field Help Text dialog box, 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 selected if the value of `<population>` is greater than 10,000. Enter the following in the Help text field: `<?population>10000?>`
5. Note that you do not have to construct an `if` statement. The expression is treated as an `if` statement.

## Using More Advanced Template Features (continued)

### Drop-Down Lists

Oracle BI Publisher enables 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, perform the following steps:

1. Position the cursor in your template where you want the value from the drop-down list to appear, and select the Drop-Down Form Field from the Forms toolbar.
2. Right-click the field to display the Drop-Down Form Field Options dialog box.
3. Add each value to the "Drop-down item" field and then click Add to add it to the "Items in drop-down list" group. The values are indexed starting from 1 for the first. For example, the list of continents is stored as follows:

| Index | Value |
|---|---|
| 1 | Asia |
| 2 | North America |
| 3 | South America |
| 4 | Europe |
| 5 | Africa |
| 6 | Australia |

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 `<?continentIndex?>`.

# Some More Advanced Template Features

- Page-level calculations
- Conditional formats
- Parameters and variables
- Data handling:
  - Grouping
  - Sorting







**Some More Advanced Template Features**

These features are discussed in detail on the following pages.

# Conditional Formatting

- Oracle BI Publisher supports the use of the following types of conditional formatting using XSL/XSL:FO code:
  - `if` statements
  - `if` statements in boilerplate text
  - `if-then-else` statements
  - `choose` statements
  - Column formatting
  - Row formatting
  - Conditionally displaying a row
  - Cell highlighting
- Oracle BI Publisher Desktop provides a wizard for defining conditional formats:
  - Insert > Conditional Format







## Conditional Formatting

Conditional formatting occurs when a formatting element appears only when a certain condition is met. Oracle 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 reported numbers reaching a certain threshold be displayed in red. Alternatively, you can use this feature to hide table columns or rows depending on the incoming XML data.

**Note:** The current version of BI Publisher Desktop also provides you with a wizard to insert conditional formats in RTF templates. Select Insert > Conditional Format.

### `if` Statements

Use an `if` statement to define a simple condition. If a data field is a specific value:

- Insert the following syntax to designate the beginning of the conditional area: `<?if:condition?>`
- 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='COMPANYA'?>` before the Supplier field on the template. Enter the `<?end if?>` tag after the invoices table. You can insert the syntax in form fields or directly into the template.

## Conditional Formatting (continued)

### `if` Statements in Boilerplate Text

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

```
The program was (not) successful.
```

You want the word `not` to appear only if the value of an XML tag called `<SUCCESS>` equals `N`. To meet this requirement, you must use the Oracle 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 about context commands, see "Using Context Commands" in the *Oracle BI Publisher User's Guide*.

For example, suppose that you construct the code as follows:

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

The following undesirable result occurs:

```
The program was
not
successful.
```

This happens because Oracle BI Publisher applies the instructions to the block by default. To specify that the `if` statement must be inserted into the inline sequence, enter:

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

This construction results in one of two displays. If `SUCCESS` does not equal `'N'`, the display is `The program was successful`. If `SUCCESS` equals `'N'`, the display is `The program was not successful`.

### `if-then-else` Statements

Oracle 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?>`

## Conditional Formatting (continued)

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 equals 1000, show Equal:

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

### choose Statements

Use the choose, when, and otherwise elements to express multiple conditional tests. If certain conditions are met in the incoming XML data, specific sections of the template are rendered. This is a very powerful feature of the RTF template. In regular XSL programming, if a condition is met in the choose command, further XSL code is executed. Use the following syntax for these elements:

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

### Column Formatting

You can conditionally show and hide columns of data in your document output.

### Row Formatting

With Oracle BI Publisher, you can 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*?> and <?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.

## Conditional Formatting (continued)

### Cell Highlighting

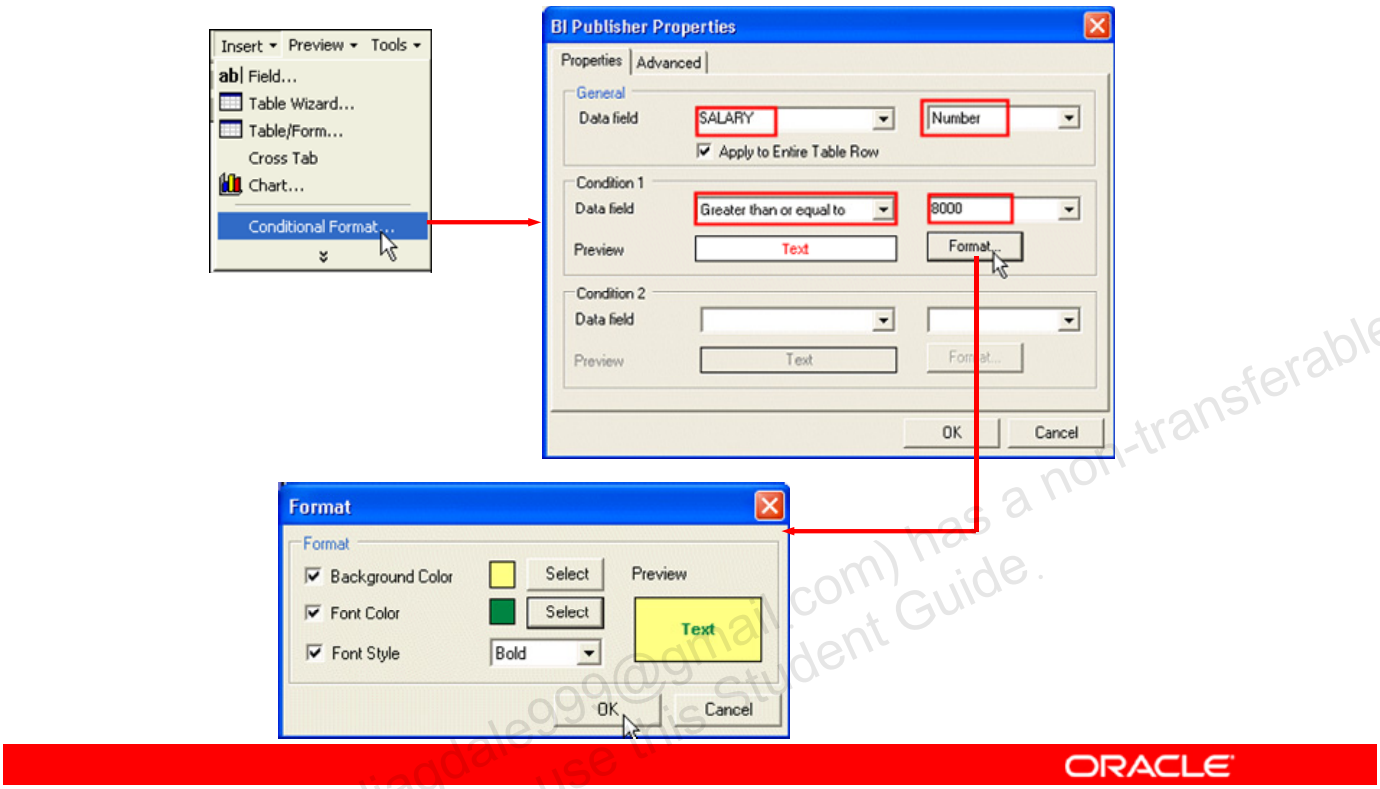
The code to highlight the debit column as shown in the table is:

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

The `if` statement tests whether the debit value is greater than 1,000. If it is, the next lines are invoked. Notice that the example embeds native XSL code inside the `if` statement. Using the `attribute` element, you can modify properties in the XSL. The `xdofo:ctx` component is an Oracle BI Publisher feature that enables 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 attribute, you can use either the standard HTML names (for example, `red`, `white`, and `green`), or you can use the hexadecimal color definition (for example, `#FFFFF`).

**Note:** Examples of conditional formatting can be found in the *BI Publisher User's Guide*.

# Conditional Formats in BI Publisher Desktop






## Conditional Formats in BI Publisher Desktop

Using the BI Publisher Desktop Insert > Conditional Format option, you can insert conditional formats in RTF templates. BI Publisher provides this feature through its simple-to-use UI that does not require the user to code. For example, in the employee salary report, you may want to show the salaries of all employees who earn more than 8,000 USD per month in green, and the salaries of all employees who are making less than 3,500 USD in red. You can do this in BI Publisher Desktop. (More details about how to create conditional formats in BI Publisher Desktop are given in the self-guided Practice 6-1).

# Page-Level Calculations

- Oracle BI Publisher supports the use of the following types of page-level calculations:
  - Page totals
  - Brought-forward or carried-forward totals
  - Running totals
- These features are performed by the PDF-formatting layer. Therefore, they are not available for other outputs types such as HTML, RTF, and Excel.







## Page-Level Calculations

### Displaying Page Totals

With Oracle BI Publisher, you can 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 such as HTML, RTF, Excel, or PowerPoint.

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

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

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

Where `TotalFieldName` is the name that you assign to your total (to refer to later), and `element` is the XML element field to be totaled.

## Page-Level Calculations (continued)

You can add this syntax to as many fields as you want to total. Then, when you want to display the total field, enter the following syntax:

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

Where *TotalFieldName* is the name you assigned to the page total field, and `number-format` is the format you want to use for the display.

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

### Brought-Forward and 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 and 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 output types, such as HTML, RTF, and Excel.

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

```
<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 in the `brought-forward` syntax:

| Code Element | Description and Usage |
|---|---|
| `inline-total` | This element has two properties:<br>`name:` Name of the variable that you declared for the field<br>`display condition:` This sets the display condition. This is an optional property that takes one of the following values:<br>`first:` Contents appear only on the first page.<br>`last:` Contents appear only on the last page.<br>`exceptfirst:` Contents appear on all pages except the first.<br>`exceptlast:` Contents appear on all pages except the last.<br>`everytime:` (default) Contents appear on every page.<br>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` | This shows the value on the page. It has the following two properties:<br>`name:` Name of the field to show—in this case, `InvAmt`. This property is mandatory.<br>`format:` The Oracle number format to apply to the value at run time. This property is optional. |

For more information, see "Number Formats" and "Date Fields" in the *Oracle BI Publisher User's Guide*.

## Page-Level Calculations (continued)

Insert the brought-forward object at the top of the template where you want the brought-forward total to appear. 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 appear in the header, you must insert the full code string into the header because Microsoft Word does not support form fields in header or footer regions.

However, you can alternatively use the Header/Footer Template technique, which enables you to enter your header and footer content within the body of the template and then simply include a call for this content in the header or footer region. This can simplify the look of your template if you have many or complex objects contained in your header or footer regions.

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

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

Note the following differences with the brought-forward object:

- The `display condition` is set to `exceptlast` so that the carried-forward total is displayed 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`.

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

**Running Totals**

To create the Running Total field, define a variable to track the total and initialize it to 0.

# Data Handling

Oracle BI Publisher supports the use of the following types of data handling:

- Sorting
- Grouping







## 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?>
```

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

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

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

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

### Grouping the XML Data

The RTF template supports the XSL 2.0 for-each-group standard that enables you to group 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 thus no longer limited by the structure of your data source.

# Variables, Parameters, and Properties

Oracle BI Publisher supports the use of the following features:

- Using variables
- Defining parameters
- Setting configuration properties in a template







## Variables, Parameters, and Properties

### Using Variables

Updatable variables differ from standard XSL variables `<xsl:variable>` in that they are updatable during the template application to the XML data. This allows you to create many new features in your templates that require updatable variables. The variables use a "set and get" approach for assigning, updating, and retrieving values. Use the following syntax to declare or set a variable value:

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

Use the following syntax to retrieve a variable value:

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

You can use this method to perform calculations, as in the following example:

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

This sets the value of variable `'x'` to its original value plus 1, much like using `"x = x + 1"`. `$_XDOCTX` specifies the global document context for the variables. In a multithreaded environment, many transformations may be occurring at the same time, and thus the variable must be assigned to one transformation.

**Note:** Handling parameters and hyperlinks is discussed in more detail in the lesson titled "Creating Parameters, List of Values (LOVs), and Hyperlinks."

## Variables, Parameters, and Properties (continued)

### Defining Parameters

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

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

### Using a Parameter in a Template

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

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

  Where *PARAMETERNAME* is the name of the parameter, `DEFAULT` is the default value for the parameter (the select statement is optional), and `xdofo:ctx="begin"` is a required string to push the parameter declaration to the top of the template at run time 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.
- Refer to the parameter in the template by prefixing the name with a `"$"` character. For example, if you declare the parameter name to be `"InvThresh"`, then reference the value by using `"$InvThresh"`.
- At run time, pass the parameter to the Oracle BI Publisher engine programmatically. Before calling either the FOProcessor API (Core) or the TemplateHelper API (E-Business Suite), create a Properties class and assign a property to it for the parameter value as follows:

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

For more information, see “Calling Oracle BI Publisher APIs” in the *BI Publisher User’s Guide*.

## Variables, Parameters, and Properties (continued)

### Setting Properties

Oracle BI Publisher properties that are available in the Oracle BI Publisher Configuration file can alternatively be embedded into the RTF template. The properties set in the template are resolved at run time by the Oracle 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 "Administration" in the *BI Publisher User's Guide* for more information about the Oracle BI Publisher configuration file and the available properties.

.

# Advanced Design Options

Oracle BI Publisher supports the use of the following advanced report layout features:

- Batch reports
- Cross-tab support
- Dynamic data columns
- Defining columns to repeat across pages
- Number and date formatting
- Calendar and time-zone support
- Using external fonts







## Advanced Design Options

If you have more complex design requirements, Oracle BI Publisher supports the use of XSL and XSL-FO elements, and has also extended a set of SQL functions. RTF templates offer extremely powerful layout options by using Oracle BI Publisher's syntax. However, because the underlying technology is based on open W3C standards, such as XSL and XPATH, you are not limited by the functionality described in this course. You can fully use the layout and data manipulation features available in these technologies.

**Note:** For inserting cross-tabs, BI Publisher Desktop provides a cross-tab wizard, which can be invoked by selecting Insert > Cross tab.

A few of these options are described on the following pages. For more information about these options, refer to the *Oracle BI Publisher User's Guide*.

## Advanced Design Options (continued)

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

BI Publisher supports this requirement through the use of a context command. Using this command, you can 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, 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?>
```

Here, `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 is as follows:

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

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

**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 correctly renders 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 run time.

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 `<?split-column-width:`*`name`*`?>` syntax, 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 `<?split-column-width:@width?>` syntax to use the value of that attribute.
- **Dynamic Column Width's unit value (in points):** `<?split-column-width-unit:`*`value`*`?>`

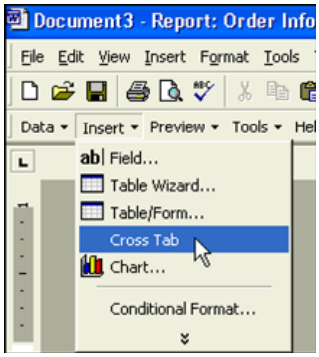
## Advanced Design Options (continued)

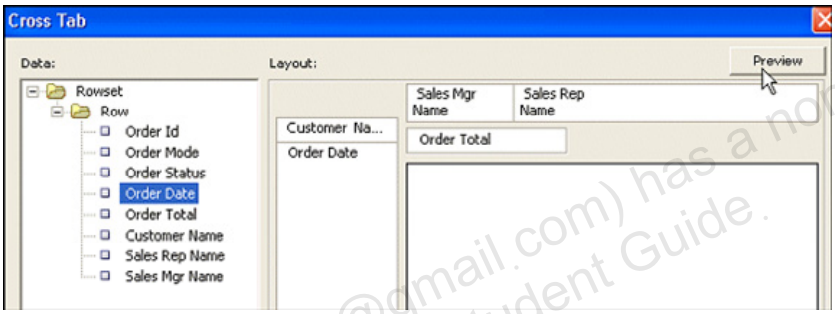
**Cross-Tab Support:** The columns of a cross-tab report are data dependent. At design time, you do not know how many columns will be reported or what the appropriate column headings will be. Moreover, if the columns should break onto a second page, you must be able to define the row label columns to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

This example uses the following XML sample:

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

# Cross-Tab Wizard in BI Publisher Desktop






## Cross-Tab Support in BI Publisher Desktop

Cross-tab reports enable you to perform your analysis on multiple facets—for example, Sales Revenues by Year and Region, Profits by Product and year, and Customer order by sales manager. These have data fields (Sales revenues, profits, order totals, and so on), row fields (product name), and column fields (year).

BI Publisher provides a wizard for inserting cross-tab reports in your RTF templates. You can select Insert > Cross Tab. This displays the cross-tab report layout so that you can easily drag the fields in your data set to create a cross-tab. (The steps are covered in more detail in the self-guided Practice 6-9).

# Summary

In this module, you should have learned how to describe and use advanced RTF template techniques.

# Practice 6: Overview

This practice covers the topics covered in the lesson:

- Creating RTF templates and using MS Word native features
- Using advanced features

Additional challenge practices are provided to give you more hands-on experience with RTF template advanced features.

ORACLE

7

# Working with PDF and eText Templates

# Objectives

After completing this lesson, you should be able to describe the following features of Oracle BI Publisher:

- The basics of PDF templates
- The advanced capabilities of PDF templates
- The basics of eText templates

# PDF Template Overview

You can take any existing PDF document and apply Oracle BI Publisher markup. You can obtain the initial source PDF document in the following ways:

- Design the layout of your template by using any application that generates documents that are convertible to PDF.
- Print the document by using Adobe Acrobat Distiller.
- Scan a paper document to PDF.
- Download a PDF document from a third-party Web site.







## PDF Template Overview

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

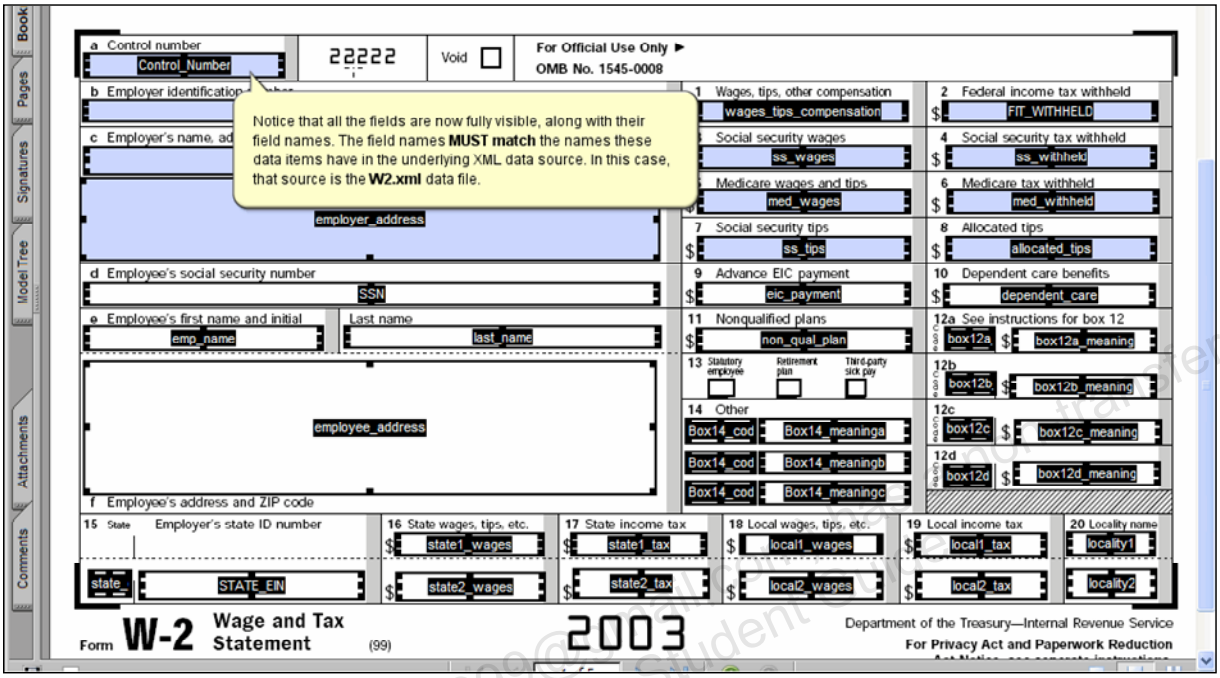
For this reason, the PDF template is not recommended for documents that require frequent updates to the layout. However, it is appropriate for forms that have a fixed layout such as invoices or purchase orders.

**Note:** Further information about creating PDF and eText templates can be found in “Creating a PDF Template” of the *BI Publisher User’s Guide*.

# Supported Modes

- Oracle BI Publisher supports Adobe Acrobat 5.0 (PDF Specification 1.4).
- If you are using Adobe Acrobat 6.0 (or later), use the Reduce File Size option to save the file so that it is compatible with Adobe Acrobat 5.0.

# Adding Markup to the Template Layout for Adobe Acrobat Users






## Adding Markup to the Template Layout for Adobe Acrobat Users

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

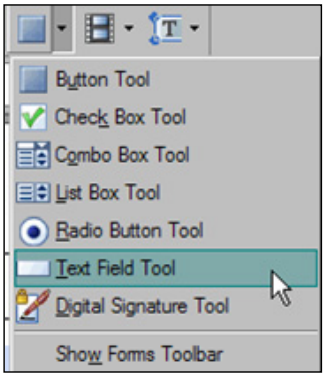
When you draw form fields in Adobe Acrobat, you are drawing them on top of the layout that you designed. There is no relationship between the design elements on your template and the form fields. Therefore, you must place the fields exactly where you want the data to display on the template.

The markup procedures for Adobe Acrobat 5.0 and 6.0 (or later) are slightly different. These procedures are outlined on the following pages.

### Example

The example in the slide shows a completely marked-up PDF template for a W2 form.

# Accessing the Text Field Tool in Adobe Acrobat






ORACLE



## Accessing the Text Field Tool in Adobe Acrobat

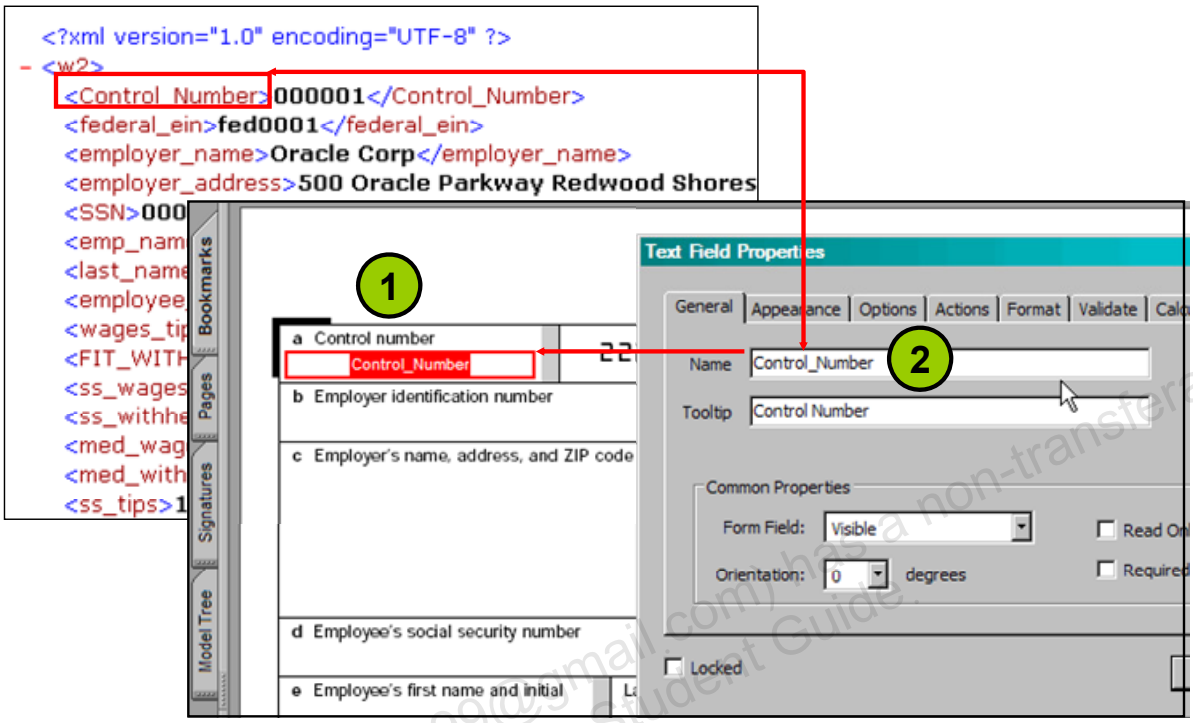
**Adobe Acrobat 6.0 (or later) (shown in the slide)**

- Select **Text Field Tool** from the Forms toolbar.

**Adobe Acrobat 5.0**

- Select **Form Tool** from the toolbar.

# Creating a Text Field in Adobe Acrobat






## Creating a Text Field in Adobe Acrobat

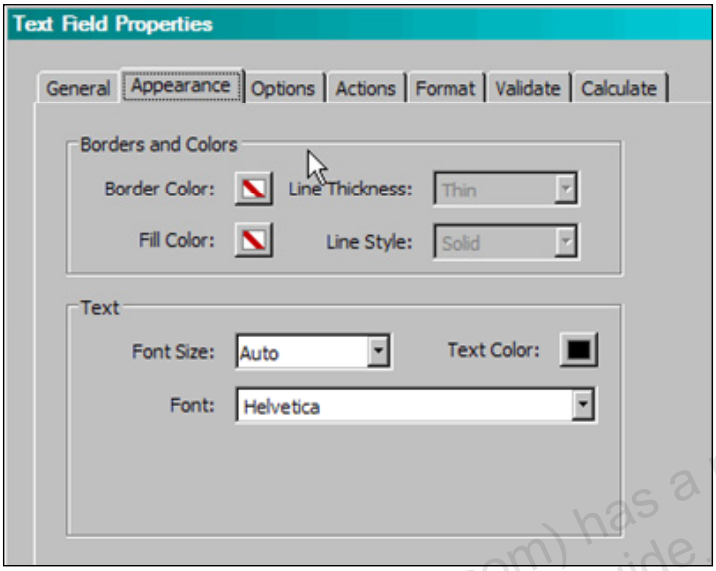
### Adobe Acrobat 6.0 (or later)

1. Draw a form field box on the template where you want the field to be displayed. Result: The Text Field Properties dialog box appears.
2. On the General tab, enter a name for the placeholder in the Name field, which must exactly match your XML source (for example, `Control_Number`).
3. Use the Text Field Properties dialog box to set other attributes for the placeholder, such as enforce maximum character size, field data type, data type validation, visibility, and formatting.
   **Note:** In the example, the PDF template is `w2.pdf` and the XML source file is `w2.XML`. Both files are part of the BI Publisher Desktop sample files.

### Adobe Acrobat 5.0

1. Draw a form field box on the template where you want the field to be displayed.
2. In the Name field of the Field Properties dialog box, enter the placeholder name, which must match the XML source field name.
3. Select **Text** from the Type drop-down list.
4. Use the Field Properties dialog box to set other attributes in the same way as with Acrobat 6.0.

# Supported Field Properties Options






## Supported Field Properties Options

BI Publisher supports the following options that are available from the Text Field Properties dialog box (in Acrobat 6.0 or later):

**General tab**

- Common properties: Read-only, required, visible/hidden, orientation (in degrees)

**Appearance tab**

- Border settings: Color, background, width, style
- Text settings: Color, font, size
- Border style

**Options tab**

- Multi-line
- Scrolling text

**Format tab:** Number category options only

**Calculate tab:** All calculation functions

**Note:** For more information about these options, see the Adobe Acrobat documentation.

# Creating a Check Box

A check box is used to present options from which more than one option 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.”







## Creating a Check Box

A certain form contains a check box listing of automobile options such as Power Windows, Sunroof, and Alloy Wheels. Each option represents a different element from the XML file. If the XML file contains a value of "Y" for a field, you want the check box to display as “checked.” All or none of these options may be selected. To create the check box, perform the following steps:

**Acrobat 6.0 (and later)**

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. Click 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 5.0**

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. Click 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.








## Creating a Radio Button Group

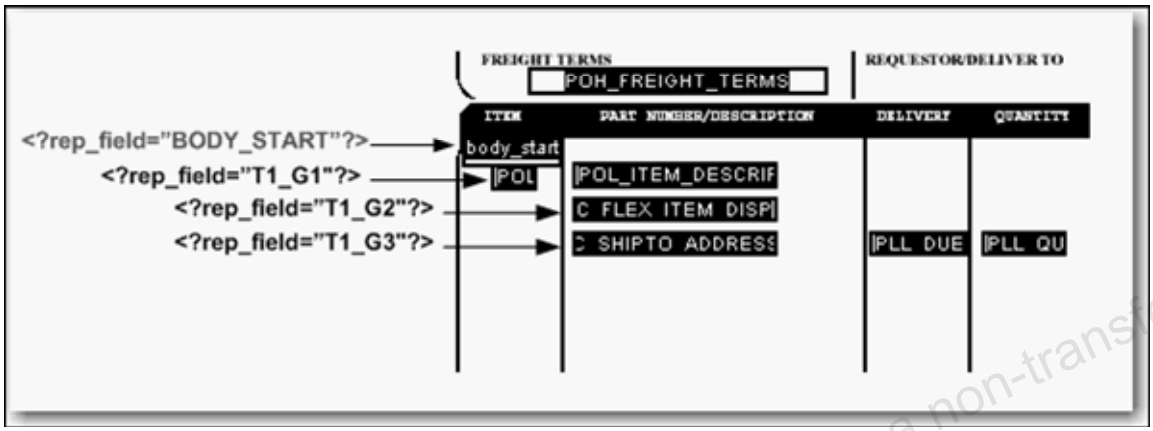
**Acrobat 6.0 (and later):**

1. Select the Radio Button tool from the Forms toolbar.
2. Draw the form field on the template.
3. On the General tab of the Radio Button Properties dialog box, enter a name for the field. Each radio button can be named differently, but must be mapped to the same XML data field.
4. Click the Options tab.
5. In the Export Value field, enter the value that the XML data field should match to enable the “on” state.

**Acrobat 5.0:**

1. Draw the form field.
2. On the Field Properties dialog box, enter a name for the field. Each radio button 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. Click 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 in the slide, enter “Standard” for the field labeled “Standard” and "Overnight“ for the field labeled “Overnight.”

# Defining Groups of Repeating Fields






## Defining Groups of Repeating Fields

In the PDF template, you explicitly define the area on the page that contains the repeating fields. For example, on the purchase order template, the repeating fields must be displayed 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, perform the following steps:

1. Insert a form field at the beginning of the area that is to contain the group. (Acrobat 6.0 users must select the Text Field tool, and then draw the form field.)
2. In the Name field of the Field Properties window, enter a name. 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 that is to contain the group.
6. In the Name field of the Field Properties window, enter a name. This field is not mapped. Note that the name you assign to this field must be different from the name you assigned to the `<?rep_field="BODY_START"?>` field.

## Defining Groups of Repeating Fields (continued)

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, perform the following steps:

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

**Note:** The placement of this field in relationship to the `BODY_START` tag defines the distance between the repeating rows for each occurrence.

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

- For fields belonging to the row that begins with `POL_ITEM_DESCRIPTION`, enter `<?rep_field="T1_G1"?>`.
- For fields belonging to the row that begins with `C_FLEX_ITEM_DISP`, enter `<?rep_field="T1_G2"?>`.
- For fields belonging to the row that begins with `C_SHIP_TO_ADDRESS`, enter `<?rep_field="T1_G3"?>`.

# Adding Page Numbers

To add page numbers, define a field in the template where you want the page number to appear and then 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@.
3. Enter a starting value for the page number in the Default field. If the XML data includes a value for this field, the starting value assigned in the template is overridden. If no starting value is assigned, it defaults to 1.

# Adding Page Breaks

In your template, you can define a page break 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):

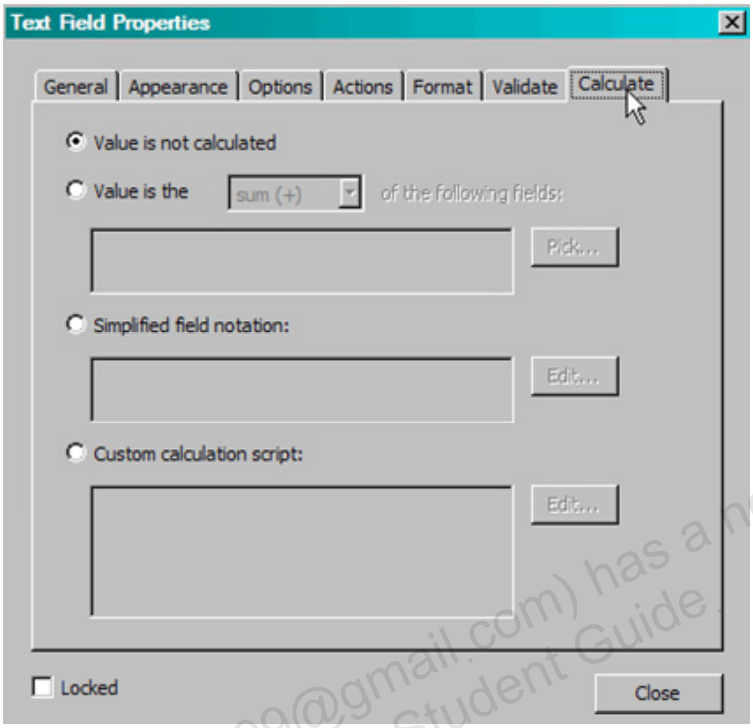
  ```
  page_break="yes"
  ```

- Example:

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

  Note: For the break to occur, the field must be populated with data from the XML source.

# Performing Calculations






## Performing Calculations

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

1. Create a text field to display the calculated total. Give the field a name.
2. In the Text Field Properties dialog box, click the Format tab and select Number from the Category list.
3. Click the Calculate tab.
4. Select the second option button (Value is the `<operation>` of the following fields:).
5. Select sum (+) from the drop-down list.
6. Click the Pick button, and select the fields that you want totaled.

# Run-Time Behavior

Placement of repeating fields:

- The placement, spacing, and alignment of fields that you create in the template are independent of the underlying form layout.
- At run time, Oracle BI Publisher places each repeating row of data according to calculations performed on the placement of the rows of created fields.







## Run-Time 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 run time, 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 is displayed exactly where you placed them on the template.

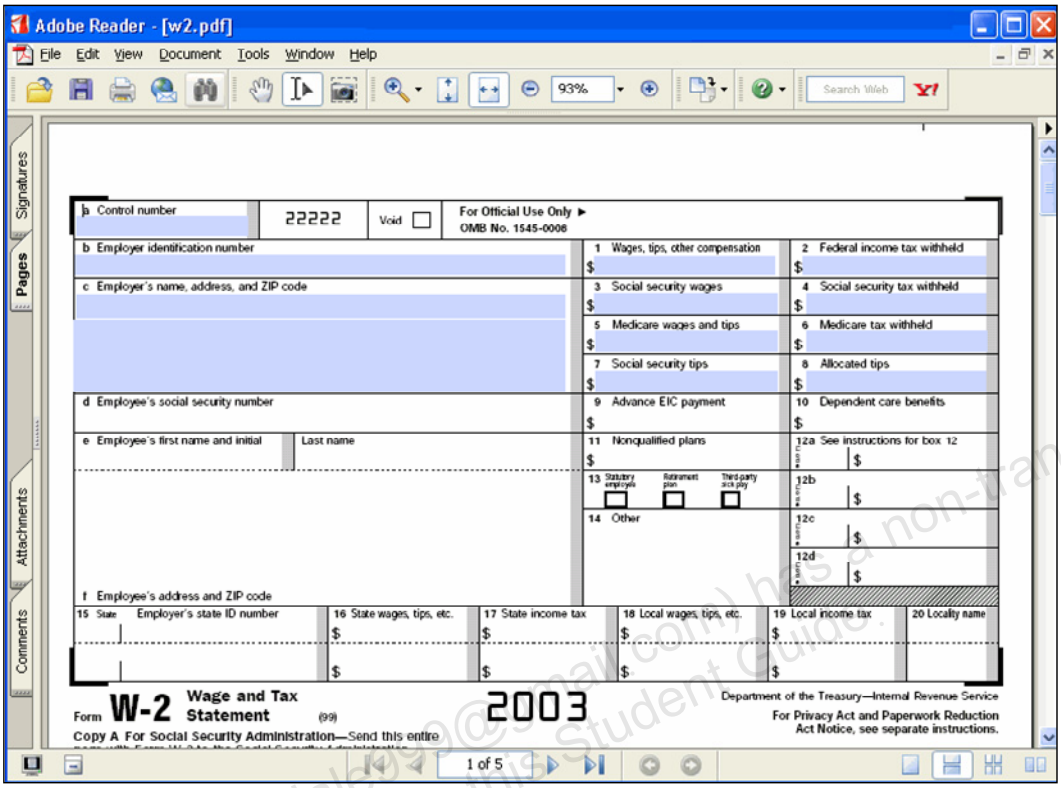
**Second occurrence, single row**

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 is placed at the calculated distance below the first occurrence.

**Second occurrence, multiple rows**

If the first group contains multiple rows, the second occurrence of the group is placed at the calculated distance below the last row of the first occurrence. The distance between the rows within the group is maintained as defined in the first occurrence.

# Downloaded PDFs




ORACLE



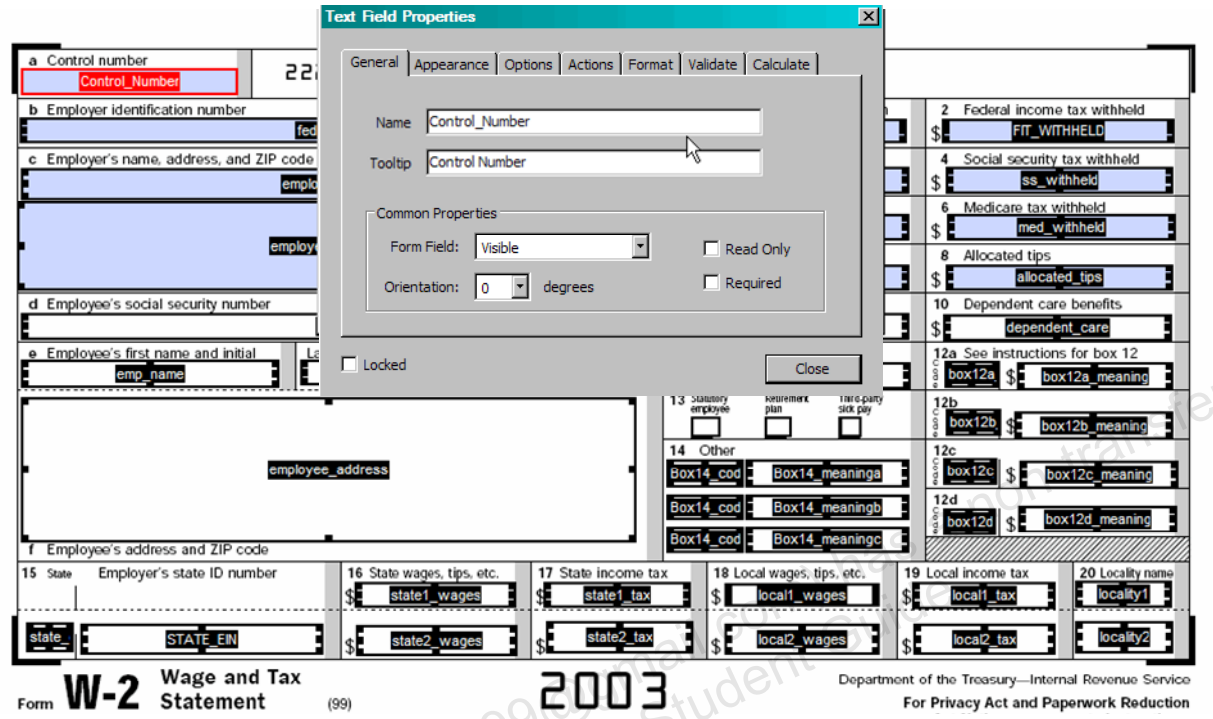
## Downloaded PDFs

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, and supply the XML data at run time to fill the report.

**Note:** Some of these forms already have form fields defined; some do not. If the form fields are not defined in the downloaded PDF, you must create them.

# Using Downloaded PDFs with Form Fields



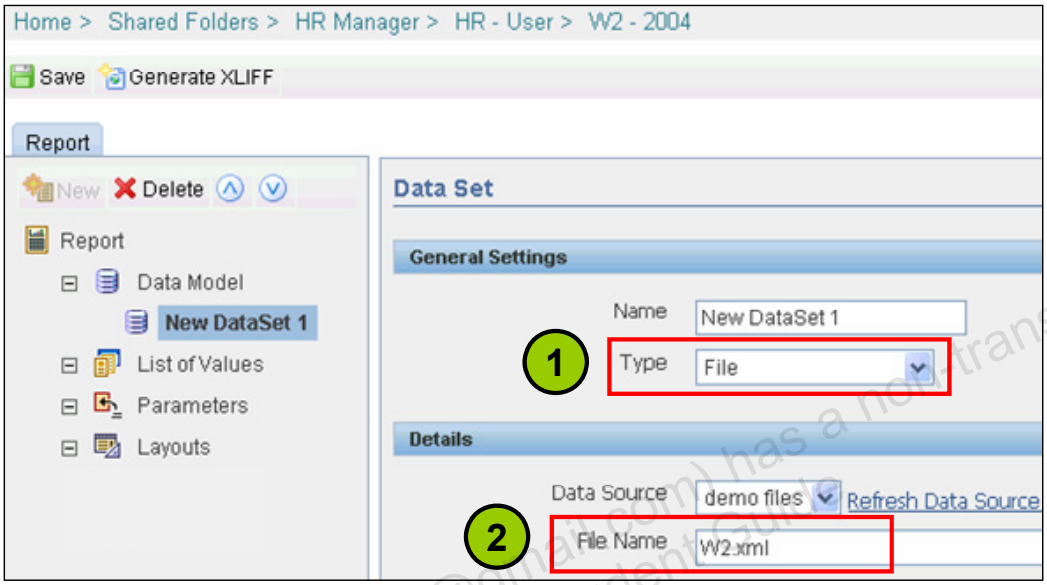



## Using Downloaded PDFs with Form Fields

The following is a summary of the steps used to prepare downloaded PDF files with form fields:

1. Download 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 highlights text fields that were already defined.
4. To map the existing form fields to the data from your incoming XML file, rename the fields to match the element names in your XML file.
5. Open the text form field's Properties dialog box either by double-clicking the field or selecting the field and then selecting Properties from the shortcut menu.
6. In the Name field, enter the element name from your input XML source.
7. Repeat steps 5 and 6 for all fields that you want populated by your data file.

# Running Reports with PDF Templates: Define Data Model






## Running Reports with PDF Templates: Define Data Model

You can run a report in BI Publisher to test your completed PDF template.

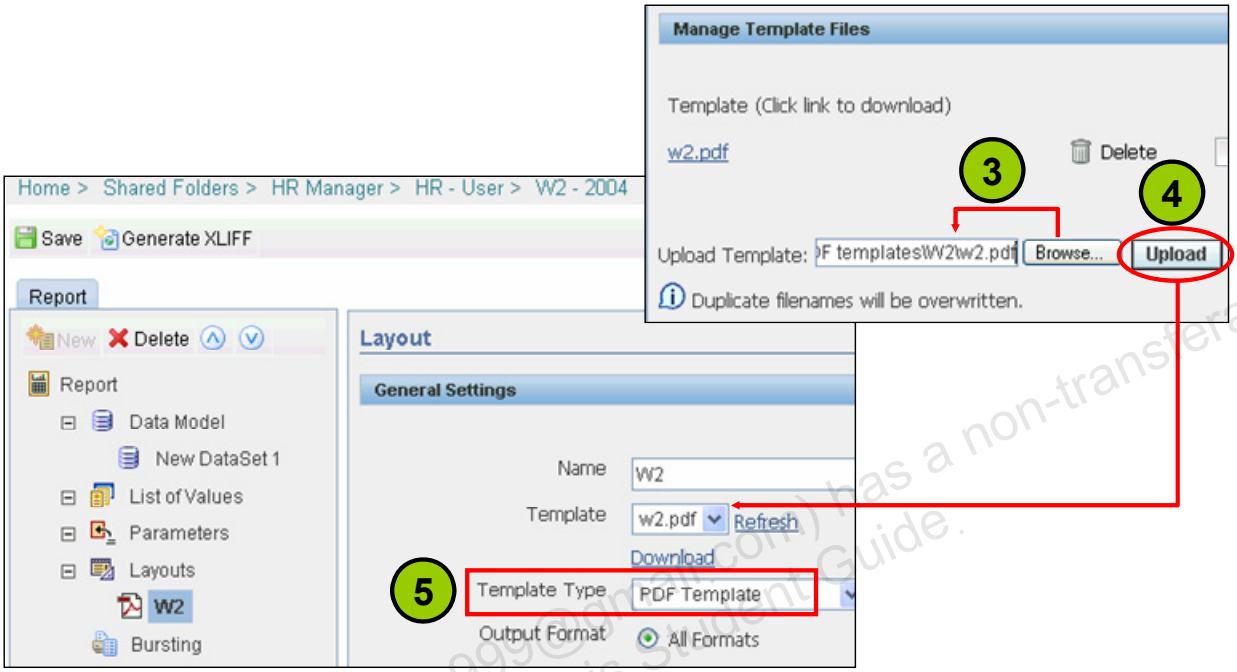
First, create a data model for the report and perform the following steps:

1. Specify File as the type.
2. Identify the name of the XML file as the source data in the File Name field.

**Example**

The data source file, `W2.xml`, is one of the sample XML data files for PDF templates.

# Running Reports with PDF Templates: Upload Template



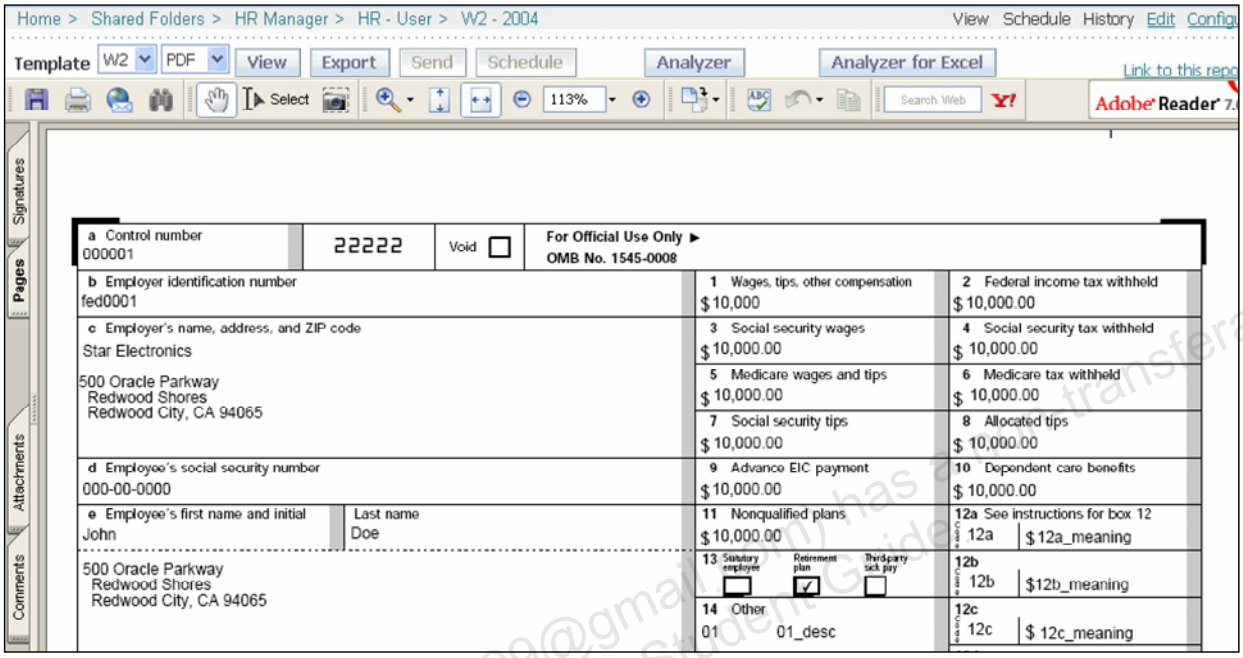



## Running Reports with PDF Templates: Upload Template

After you define the data model, create a new layout and perform the following steps:

3. In the Manage Template Files pane, click Browse to find the PDF template file.
4. Click Upload.
5. In the General Settings pane, select PDF Template as the template type.

# Running Reports with PDF Templates: View Report






## Running Reports with PDF Templates: View Report

You view the PDF template report in BI Publisher just as you would any report with an RFT template.

After the PDF template is associated with the BI Publisher report (via access to the XML file as its data source) and the report is saved, click View.

**Example**

The example shows the result of the completed `w2.pdf` template combined with the `w2.xml` data source file.

# eText Templates

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)
- Applied at run time by Oracle BI Publisher to an input XML data file to create an output text file that can be transmitted to a bank or another customer







## eText Templates

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 run time, 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 another customer. Because the output is intended for electronic communication, the eText templates must follow 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 that it is not 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 lay out on standard size paper.

## eText Templates (continued)

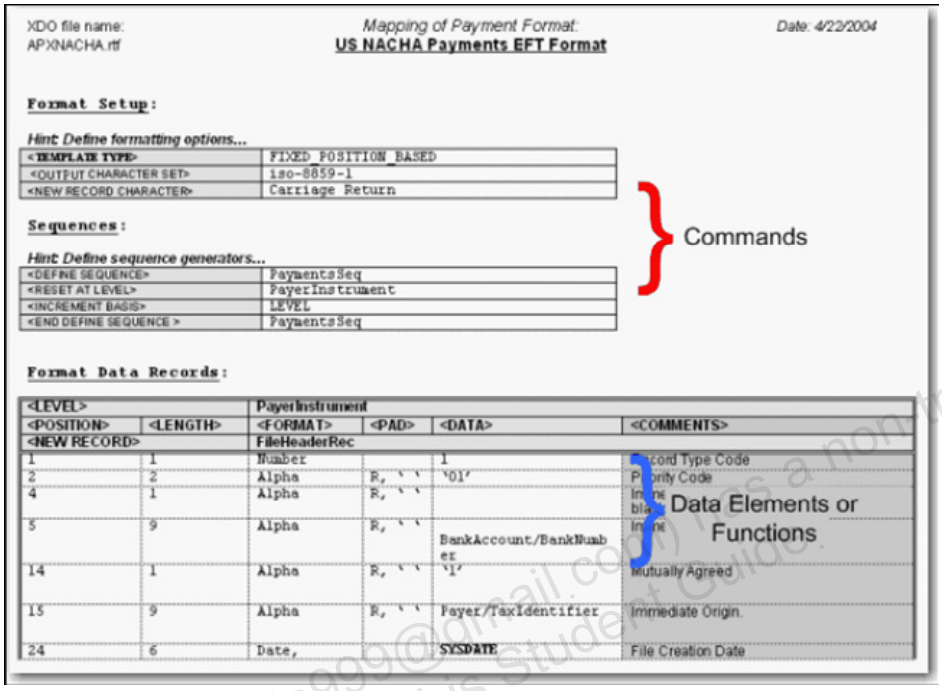
To accommodate the record length, the EFT and EDI templates are designed using tables. Each record is represented by a table. Each row in a table corresponds to a field in a record. The columns of the table specify the position, length, and value of the field.

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 run time, Oracle BI Publisher constructs the output file according to the setup commands and layout specifications in the tables.

**Note:** Further information about creating eText templates can be found in "Creating an eText Template" in the *BI Publisher User's Guide*.

# Structure of eText Templates

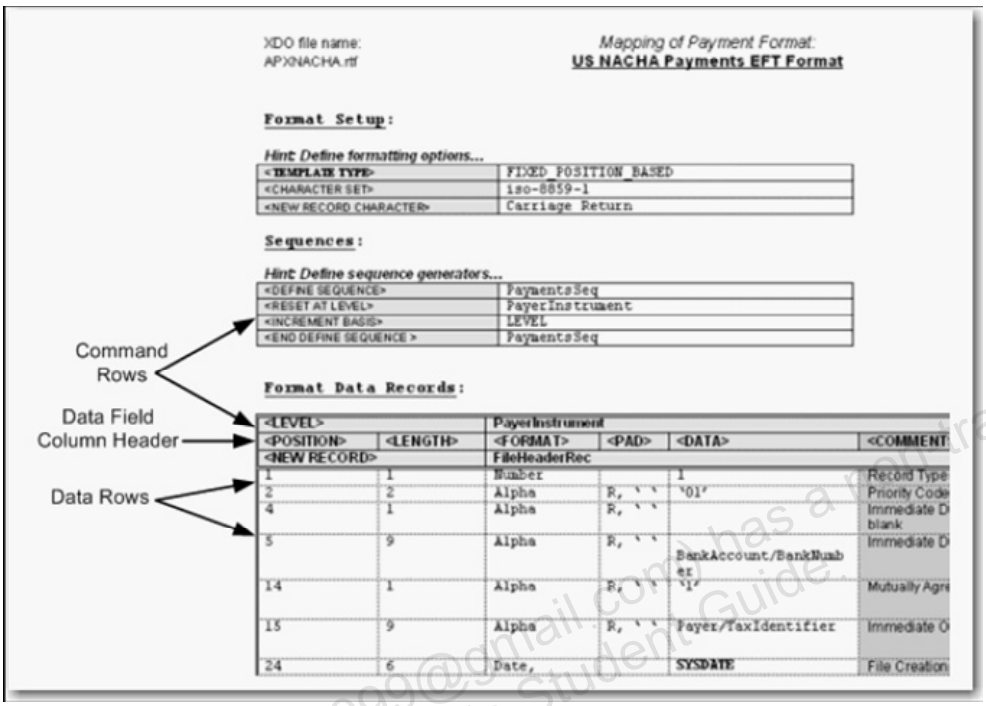





## Structure of eText Templates

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

# Row Types






## Row Types

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. 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 Oracle BI Publisher when the template is parsed.

Data column headers specify the column headings for 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 are ignored at run time.

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

# Setup Command Tables

A template always begins with a table that defines global attributes and program elements. The setup commands are:

- Template type
- Output character set
- New record character
- Invalid characters
- Replace characters
- Define level
- Define sequence
- Define concatenation







**Setup Command Tables**

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

# Constructing Data Tables

- 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

Command rows always have two columns:

- Command name
- Command parameter

The supported commands are:

- `LEVEL`
- `NEW RECORD`
- `SORT ASCENDING`
- `SORT DESCENDING`
- `DISPLAY CONDITION`







## Command Rows

### `LEVEL` Command

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

`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 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 that is defined for the parent element.

An XML element can be associated with only one level. All 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 is specified at the end of the final table.

## Command Rows (continued)

### `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 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 records in a table are at the same hierarchy level. They are printed in the order in which they are specified in the table.

### `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 want 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 must 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 applies to the rest of the fields through to 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 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







**Structure of Data Rows**

The first five columns (listed in the slide) are required and must be declared in the order stated.

The Comments column is optional and ignored by the system. You can insert additional information columns if you want; however, all columns after the required ones are ignored.

# Structure of Data Rows

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

- Maximum length
- Format
- Data
- Tag
- Comments







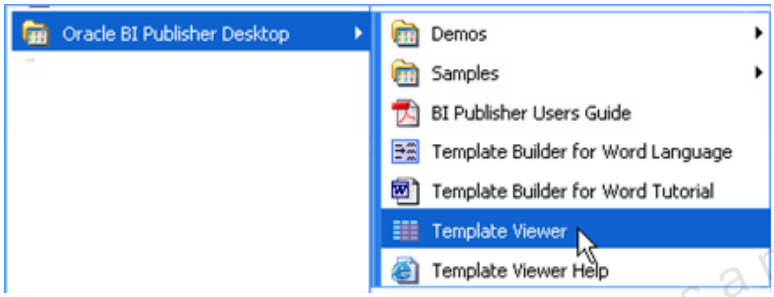
**Structure of Data Rows (continued)**

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

The last two columns are optional. The Comments column is ignored by the system. You can insert additional information columns if you want; however, all columns after the required ones are ignored.

# Using Template Viewer

You can use BI Publisher Template Viewer to view templates, forms, and style sheets.






## Using Template Viewer

BI Publisher Template Viewer is a Java application that facilitates the rapid development of format templates for use with Oracle BI Publisher. The application requires Java Runtime Environment (JRE) version 1.5.x or later installed on your desktop.

Template Viewer enables you to preview:

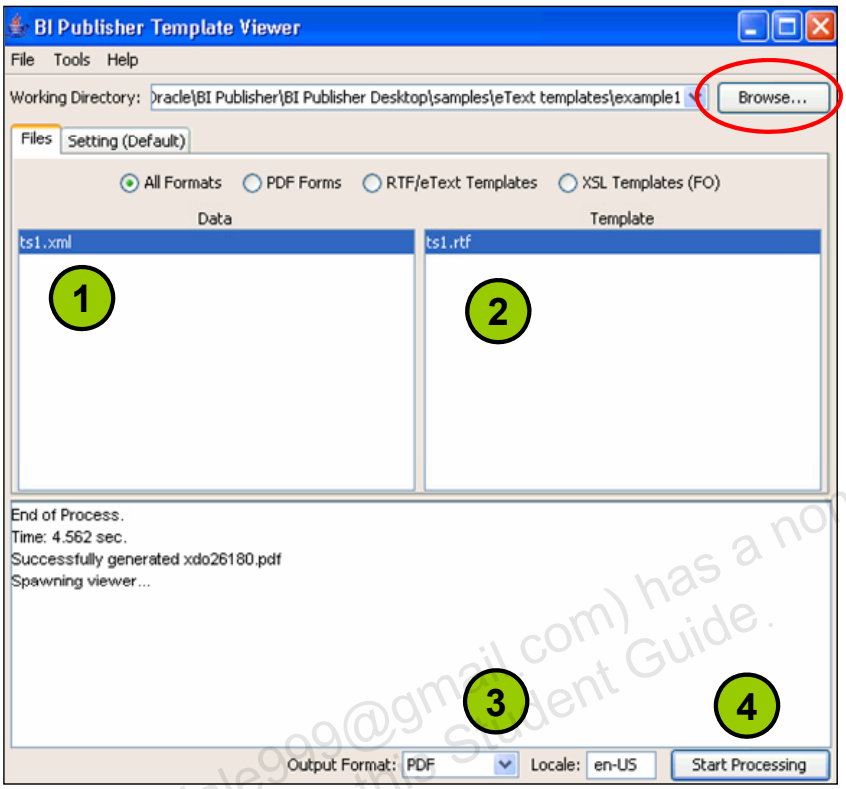
- RTF templates
- PDF forms
- eText templates
- XSL-FO style sheets

**Why Use Template Viewer?**

In general, Template Viewer is a tool that is used when using editors other than Word or when advanced tags are used.

**Note:** If you are using Word, you can view the results of your report directly in Oracle BI Publisher Desktop. But Template Viewer reproduces the same capabilities, and it has capabilities that are not available to you through Word (namely, viewing PDF templates).

# Viewing an eText Template






## Viewing an eText Template

Browse and select a working directory that contains the XML files and the corresponding template files. To generate an output, perform the following steps:

1. Select the XML file (listed on the left side).
2. Select a template (listed on right side).
3. Select an output format (such as PDF, HTML, and so on) from the drop-down list.
4. Click the Start Processing button.

The bottom pane shows information about logging or errors during the processing phase.

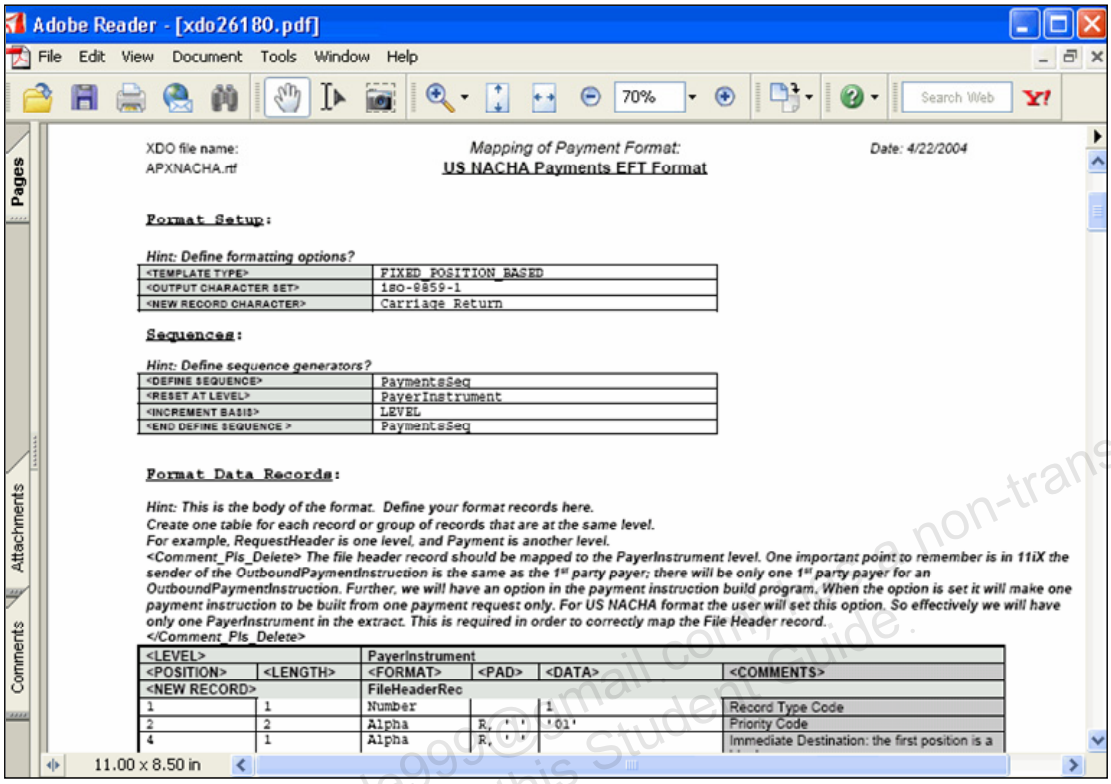
**Note:** Double-click the XML to spawn the XML in a browser. Double-click the template to start your RTF editor (for example, Microsoft Word or Adobe Acrobat, depending on the template type).

**Example**

In this example, your working directory is the following:

```
D:\Program Files\Oracle\BI Publisher Desktop\samples\eText
templates\example1\
```

# Viewing the Output






## Viewing the Output

The output is displayed in the format you chose (in the example, PDF format).

# Summarizing eText Templates

eText templates are specific to the application (EDI or EFT) and are rigidly and tightly controlled. Because of this, eText template creation is not trivial. The *Oracle BI Publisher User's Guide* must be consulted.

# Summary

In this lesson, you should have learned how to describe the following features of Oracle BI Publisher:

- The basics of PDF templates
- The advanced capabilities of PDF templates
- The basics of eText templates

8

# Creating Reports by Defining XML Data Templates



ORACLE

# Objectives

After completing this module, you should be able to do the following:

- Describe the BI Publisher XML data template
- Create an XML data template
- Associate that data template with a rich text format (RTF) template
- View the report

# XML Data Template

An XML document whose elements collectively define how the data engine processes the template to generate the XML


**Data engine process instructions**



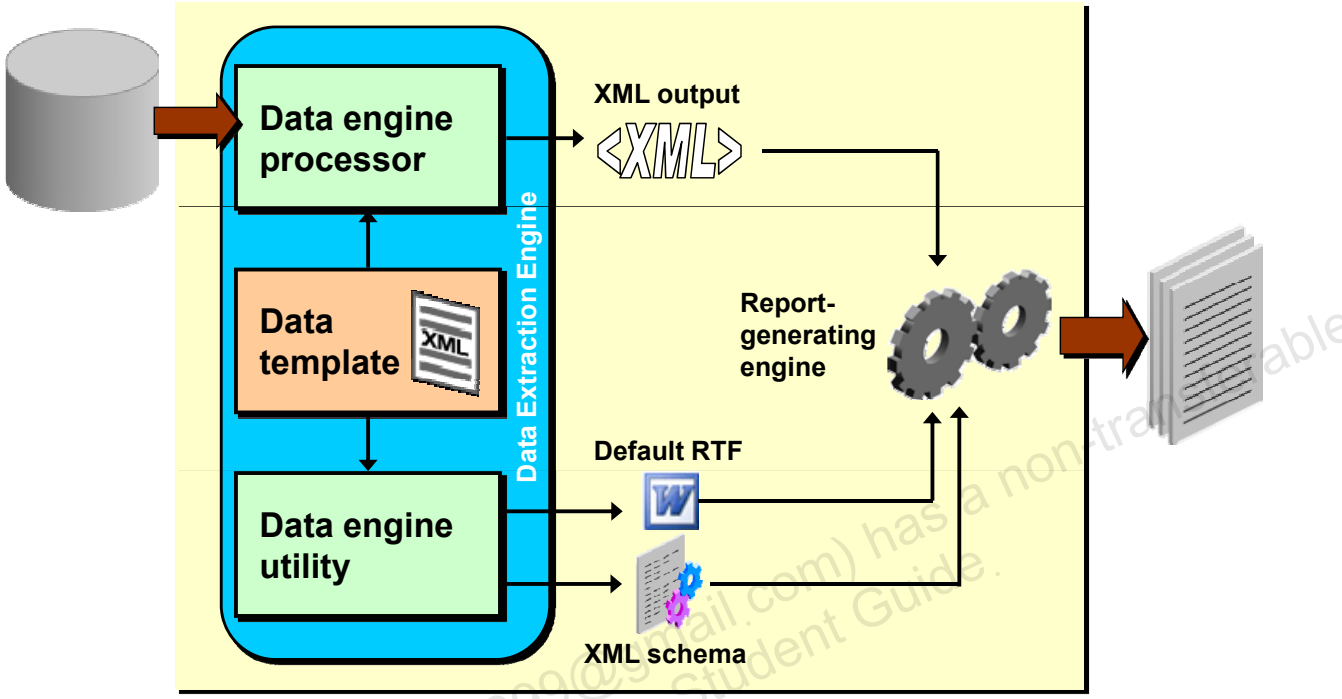




## XML Data Template

With the Oracle BI Publisher data engine, you can 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.

# Data Extraction Engine






## Data Extraction Engine

The Data Template, which defines how the data engine processes its information, is:

- Used by the data engine processor to generate XML output
- Associated with an RFT document or XML schema by the data engine utility

The XML output and RFT template / XML schema are used by the report generation engine to produce the report.

# What Functionality Is Supported?

The data engine supports the following functionality:

- Schema generation
- Default RTF template generation
- Single and multiple data queries
- Query links
- Parameters
- Aggregate functions (`SUM`, `AVG`, `MIN`, `MAX`, `COUNT`)
- Event triggers
- Multiple data groups



**What Functionality Is Supported?**

This lesson covers only the functionality that is supported in a stand-alone instance.

# Supported Functionality

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

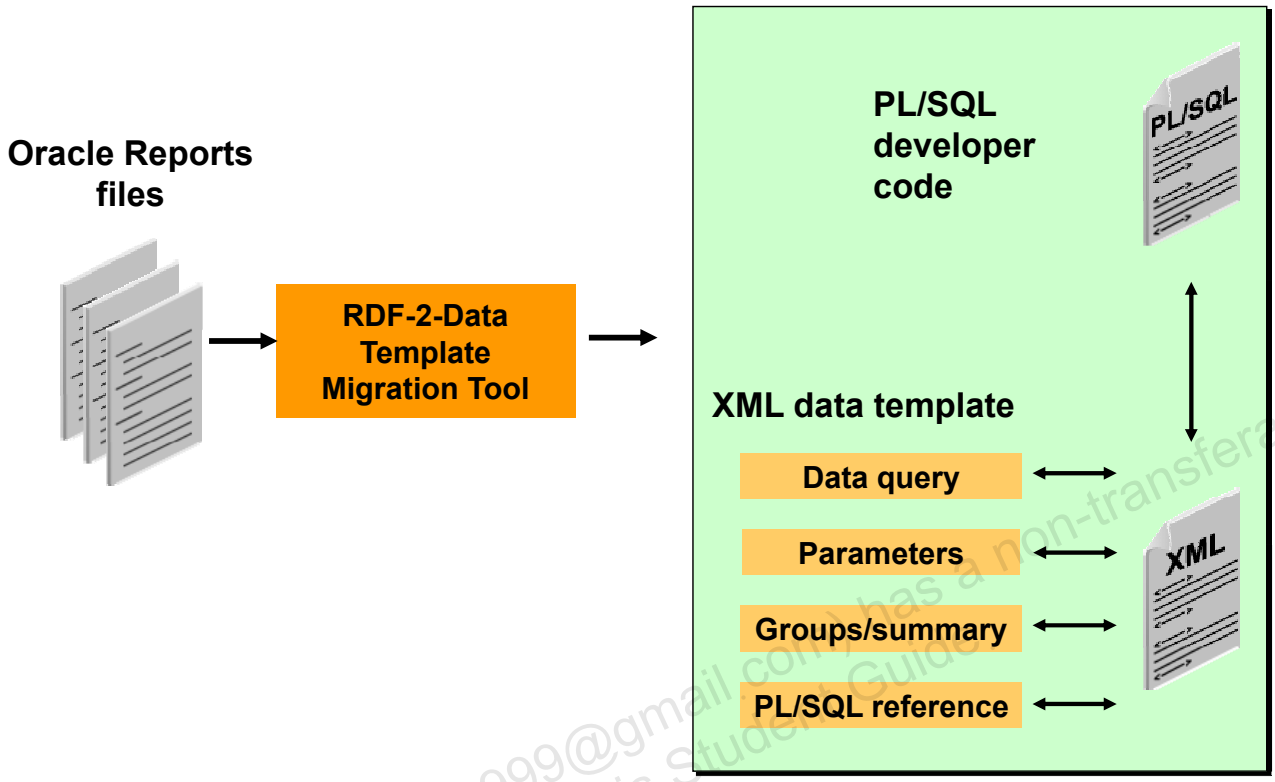
- Unicode for XML output

  Unicode is a global character set that enables multilingual text to be displayed in a single application. With this, you can develop a single multilingual application and deploy it worldwide.

- Canonical format

  The data engine generates date elements by using the canonical ISO date format: YYYY-MM-DDTHH24:MI:SS.FF3TZH:TZM for a mapped date element, and #######.## for number elements in the data template XML output.

# Report Migration






## Report Migration

Oracle Reports files can be migrated to BI Publisher by using the RDF-2 Data Template Migration Tool.

# Data Template Definition


The data template is an XML document that consists of four basic sections.



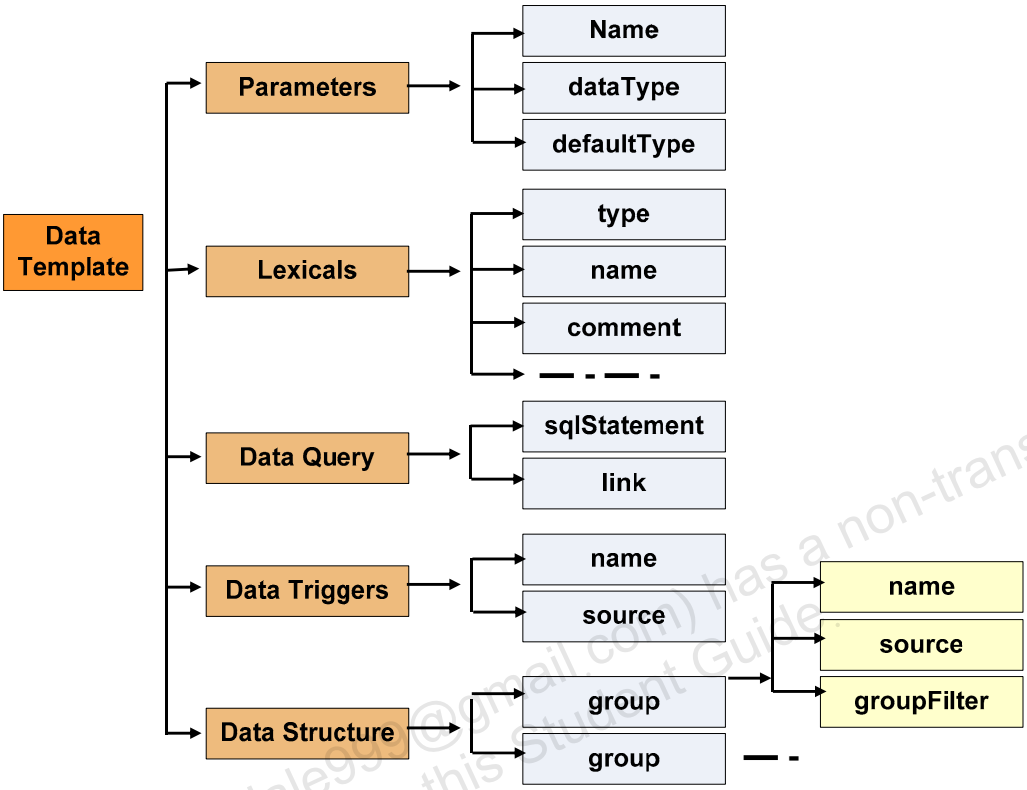
## Data Template Definition

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
- A `<dataTrigger>` section, in which trigger events are stored
- A `<dataStructure>` section in which the output XML structure is defined

See the next page for a table of elements.

# Data Template Structure Diagram










## Data Template Structure Diagram

| Element | Description |
|---|---|
| `dataTemplate` | (Required) Attributes are:<br>name (Required)<br>description<br>version (Required)<br>defaultPackage<br>dataSourceRef |
| `parameters` | Consists of one or more <parameter> elements |
| `parameter` | Attributes are:<br>name (Required)<br>dataType |
| `Lexicals` * | Consists of one or more lexical elements to support flexfields<br>* (Only used with Enterprise Business Suite) |
| `lexical` | Four types: KFF segments, KFF, KFF where, and KFF order by |
| `dataQuery` | (Required) Consists of one or more <sqlStatement> elements |
| `sqlStatement` | (Required) Attributes are:<br>name (Required)<br>dataSourceRef |

## Data Template Structure Diagram (continued)

| Element | Description |
|---|---|
| `link` | Attributes are:<br>parentQuery<br>parentColumn<br>childQuery<br>childColumn<br>condition |
| `dataTrigger` | The trigger and the event that it fires on. Attributes are:<br>name (Required)<br>source (Required) |
| `dataStructure` | Required for multiple queries group<br>Consists of one or more <element> elements and sub <group> elements. Attributes are:<br>name (Required)<br>source (Required)<br>groupFilter |
| `element` | The tag name to assign the element in the XML data output<br>Attributes are:<br>name<br>value (Required)<br>function |

# Data Template Declaration

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

| Attribute Name | Description |
|---|---|
| name | (Required) Enter the data template name. |
| description | (Optional) Enter a description of this data template. |
| version | (Required) Enter a version number for this data template. |
| defaultPackage | This attribute is required if your data template contains lexical references or any other calls to PL/SQL. |
| dataSourceRef | The default data source reference for the entire data template. Required in the following cases:<br>• XML Publisher Enterprise implementations: Always required.<br>• Oracle E-Business Suite implementations: Required only when performing a distributed query across multiple data sources. |



## Data Template Declaration

The data template element provides essential information about the template, and is the root element in the data template. It contains two required and three optional attributes.

# Defining Parameters

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

| Attribute Name | Description |
|---|---|
| name | **Required**. A keyword, unique within a given Data Template, that identifies the parameter. |
| dataType | Optional. Specify the parameter data type as "character", "date", or "number". Default value is "character". For the "date" dataType, the following three formats (based on the canonical ISO date format) are supported:<br>• YYYY-MM-DD (example: 1997-10-24)<br>• YYYY-MM-DD HH24:MI:SS (example: 1997-10-24 12:00:00)<br>• YYYY-MM-DDTHH24:MI:SS.FF3TZH:TZM |
| defaultValue | Optional. This value will be used for the parameter if no other value is supplied from the data at runtime. |



## Defining Parameters

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

### How to Define Parameters

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

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

## Defining Parameters (continued)

### How to Pass Parameters

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

**Example**

Query:

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

At run time, the value of the department is passed to the query:

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

# Defining Queries

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

| Attribute Name | Description |
|---|---|
| name | A unique identifying name for the query. Note that this name will be referred to throughout the data template. |
| dataSourceRef | (For E-Business Suite implementations only, not applicable for XML Publisher Enterprise). Specify the database against which to execute the query. If this attribute is not populated, the default data source defined in the dataTemplate element will be used. |



## Defining Queries

The `<sqlStatement>` element is placed between the open and close `dataQuery` tags. The `<sqlStatement>` element has a related attribute and name. It is expressed within the `<sqlStatement>` tag. The query is entered in the CDATA section.

**Note:** The Data Query section is mandatory.

**Example**

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

# Data Query

How to define queries:

- Use `<sqlStatement name="">` to define a query.
- Performing operations in SQL is faster than performing them in a data template or PL/SQL. The following are the most common cases in which using SQL improves performance:
  - Use a `WHERE` clause instead of a group filter to exclude records.
  - Perform calculations directly in your query rather than in the template.







**Data Query**

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, the default column name is used. This becomes important when you specify the XML output in the data structure 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.

## Data Query (continued)

### 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 run time.

Create a lexical reference by 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.

**Example**

```
Package employee
AS
           where_clause varchar2(1000);
        .....
Package body employee
AS
             .....
     where_clause := 'where deptno=10';
  .....
```

**Data template definition:**

```
<dataQuery>
  <sqlStatement name="Q1">
            <![CDATA[SELECT ENAME, SAL FROM EMP
&where_clause]]>
  </sqlStatement>
</dataQuery>
```

# Example: Data Query

```
<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
        from dept d, emp e
        where d.deptno=e.deptno
        AND d.deptno = nvl(:p_DeptNo,d.deptno)  ]]>
  </sqlStatement>
  </dataQuery>
```

# Defining a Data Link Between Queries

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

| Attribute Name | Description |
|---|---|
| name | Required. Enter a unique name for the link. |
| parentQuery | Specify the parent query name. This must be the name that you assigned to the corresponding <sqlstatement> element. |
| 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. |
| childColumn | Specify the child column name. |







## Defining a Data Link Between Queries

The `dataQuery` section can contain multiple SQL statements. If you have multiple queries, you must link them to create the appropriate data output.

# Query Linking Options

Two ways of linking are supported:

- Bind variables in your query
- Link elements `<link name="" ……../>`

Bind variables are usually more efficient than link elements.







## Query Linking Options

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. For building data queries in the data template, BI Publisher tests have shown that bind variables is more efficient than the link tag.

**Bind Variable**

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

**Link Tag**

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

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

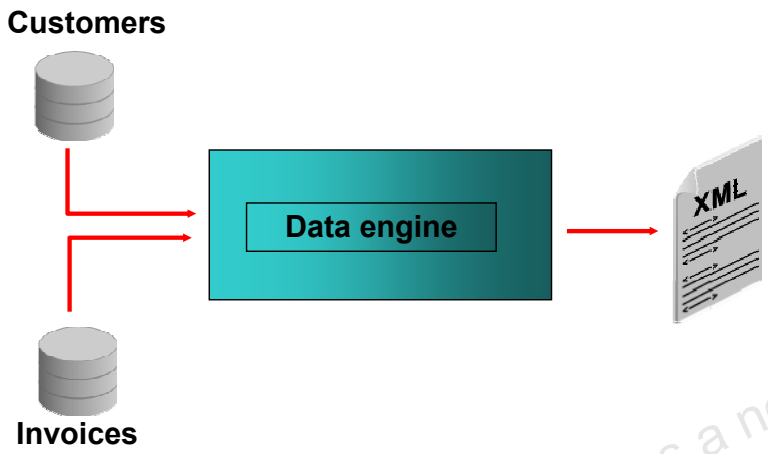
# Query Linking: Bind Variable Example

```
<dataQuery>
   <sqlStatement name="Q1">
   <![CDATA[
            SELECT DEPTNO,DNAME,LOC from dept where
    &pwhereclause
            order by deptno    ]]>
   </sqlStatement>
  <sqlStatement name="Q2">
 <![CDATA[
            SELECT  EMPNO,ENAME,JOB,MGR,HIREDATE,SAL
                     from EMP
                     WHERE DEPTNO = :DEPTNO   ]]>
  </sqlStatement>
  </dataQuery>
```

# Query Linking: <link> Tag Example

```
<dataQuery>
   <sqlStatement name="Q1">
   <![CDATA[
            SELECT DEPTNO,DNAME,LOC from dept where
&pwhereclause
            order by deptno    ]]>
   </sqlStatement>
  <sqlStatement name="Q2">
 <![CDATA[
            SELECT  EMPNO,ENAME,JOB,MGR,HIREDATE,SAL
                     from EMP ]]>
  </sqlStatement>
 <link name="DEPTEMP_LINK" parentQuery="Q1"
parentColumn="DEPTNO" childQuery="Q2"
childColumn="DEPTNO"condition="="/>
  </dataQuery>
```

# Distributed Queries






## Distributed Queries

With the BI Publisher data engine, you can perform distributed queries. This enables you to access data from multiple disparate data sources.

For example, you can extract your customer data from one database and the customer's invoice data from another system, and then bring them together into a combined hierarchical XML result set.

**Note:** The only supported method to link distributed queries is the bind variable method. (The `<link>` tag is not supported.)

For more information about creating distributed queries, see the *BI Publisher Administration and Developer Guide*.

# Data Triggers

```
<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. |







## Data Triggers

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

# Using Data Triggers

- Data triggers can be used to:
  - Perform initialization tasks
  - Build dynamic queries
- Data trigger types
  - `beforeReport` trigger: Fires before the data query is executed.
  - `afterReport` trigger: Fires after you exit and after XML output is generated.

Example:

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



## Using Data Triggers

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

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

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

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

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

# Data Structure Section

In the data structure section, you define what the XML output will be and how it will be structured. You can:

- Define a group hierarchy
- Create break groups
- Apply group filters
- Create summary columns







## 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 columns within each group and break the order of those columns; you can use summaries and placeholders to further customize within the groups. The data structure section is required for multiple queries and optional for single queries. If omitted for a single query, the data engine generates flat XML.

For more information about how to build this section, see the *BI Publisher User's Guide*.

# Data Structure: Example

```
<group name="G_DEPT" source="Q1"
groupFilter="empdata.G_DEPTFilter(:DEPT_NUMBER)">
        <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>
```

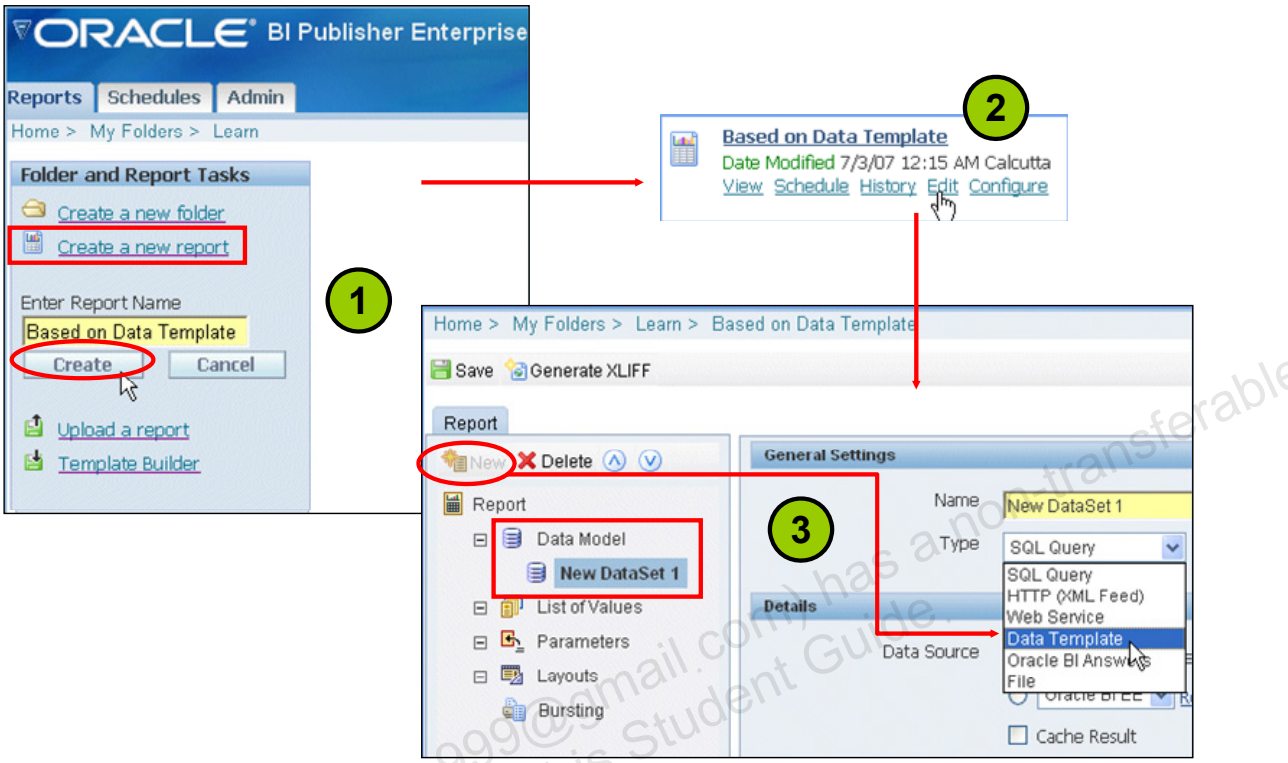


## Data Structure: Example

The data structure contains:

- **Break groups:** Order By in SQL query must be set.
- **Group filters:** WHERE clause must be used instead of a group filter.
- **Summary column:** These include the Sum, Average, Count, Maximum, and Minimum columns.

# Creating an XML Data Template





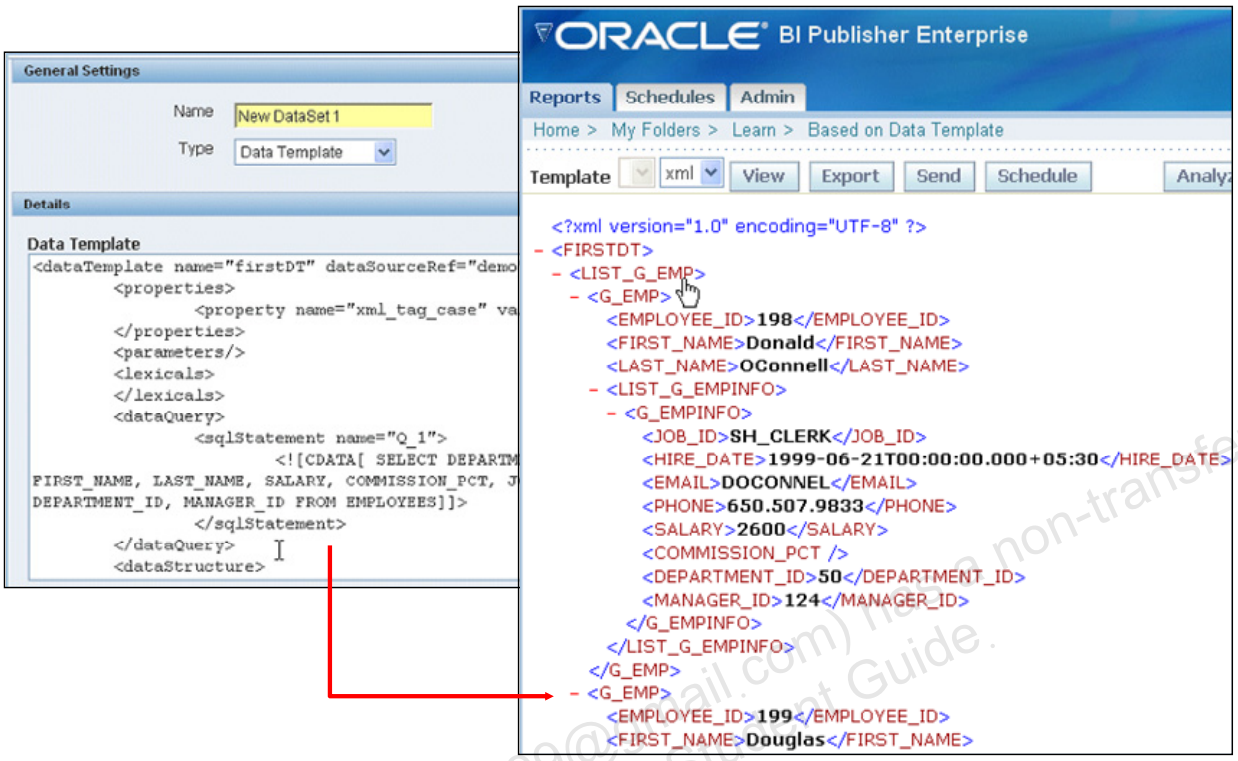

## Creating an XML Data Template

To create a data template in BI Publisher Enterprise, perform the following steps:

1. Click the "Create a new report" link in the Folder and Report Tasks pane, enter a report name (`Based on Data Template`) in the box, and click Create.
2. Click the Edit link below the name of the report created to open the Report Properties page.
3. On the Report Properties page, click the Data Model node on the left and click the New icon. In the General Settings pane that appears on the right, select Data Template from the Type drop-down list.

**Result:** The basic structure for the data template appears in the Data Template window. The XML code for your data template can be added here.

# Viewing the XML Data



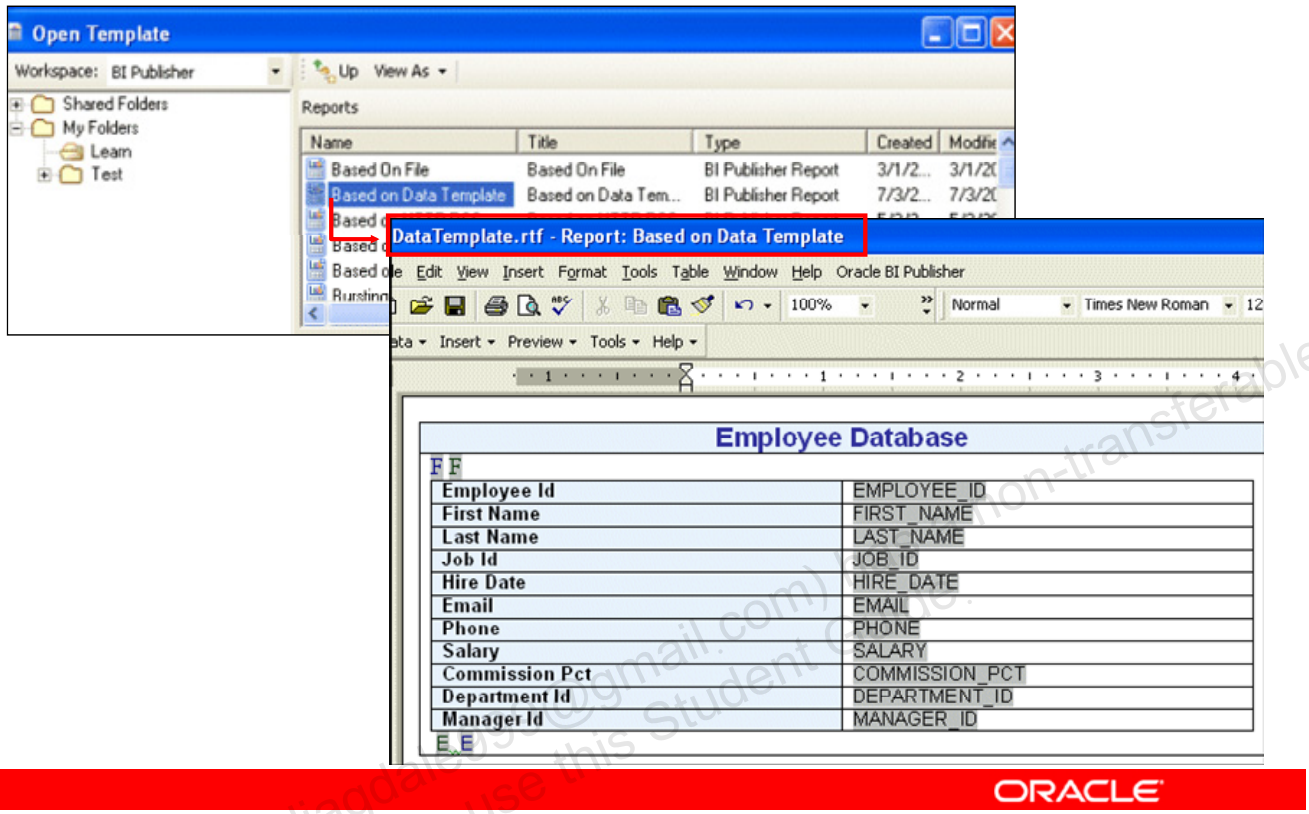

ORACLE



## Viewing the XML Data

At any time, click View to see the XML data. If the data is properly rendered to the report, the XML is correct.

# Associating the XML Data Template with an RTF Template

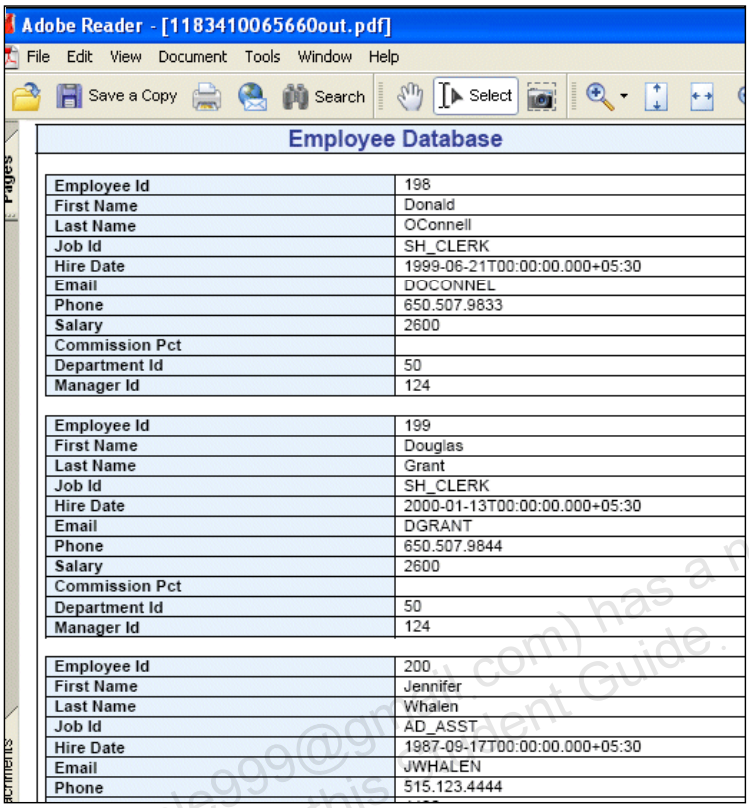





## Associating the XML Data Template with an RTF Template

Normally, you associate a BI Publisher report with an RTF template file. In this case, the XML data template is treated as a "report," and you associate it with an RTF template.

# Viewing the Report




ORACLE



## Viewing the Report

After the XML data template is associated with an RTF template, the resulting report can be viewed in BI Publisher Desktop or BI Publisher Enterprise.

# Summary

In this module, you should have learned how to:

- Describe the BI Publisher XML data template
- Create an XML data template
- Associate that data template with an RTF template
- View the report

# Practice 8: Overview

This practice covers the following topics:

- Creating a simple data template
- Creating a data template with parameters
- Creating a data template with a list of values

9

# Oracle BI Publisher: Template Creation



ORACLE

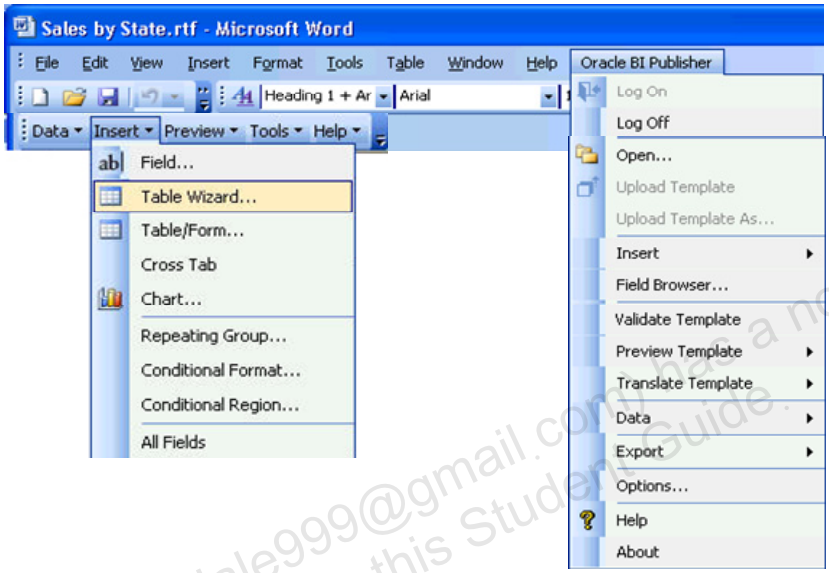
# Objectives

After completing this lesson, you should be able to:

- Create a BI Publisher report

# Creating a Template

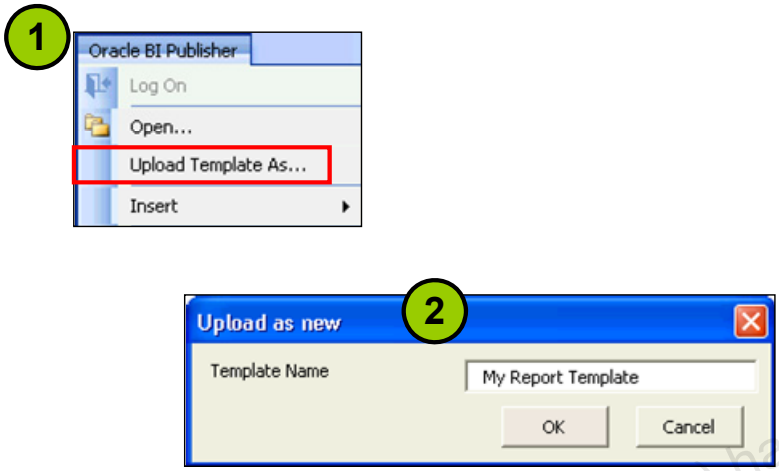
In Word, enter text, graphics, and other common word processing elements, and insert fields, charts, and tables to create a BI Publisher template.




ORACLE

# Uploading the Template

From Word, upload the template.





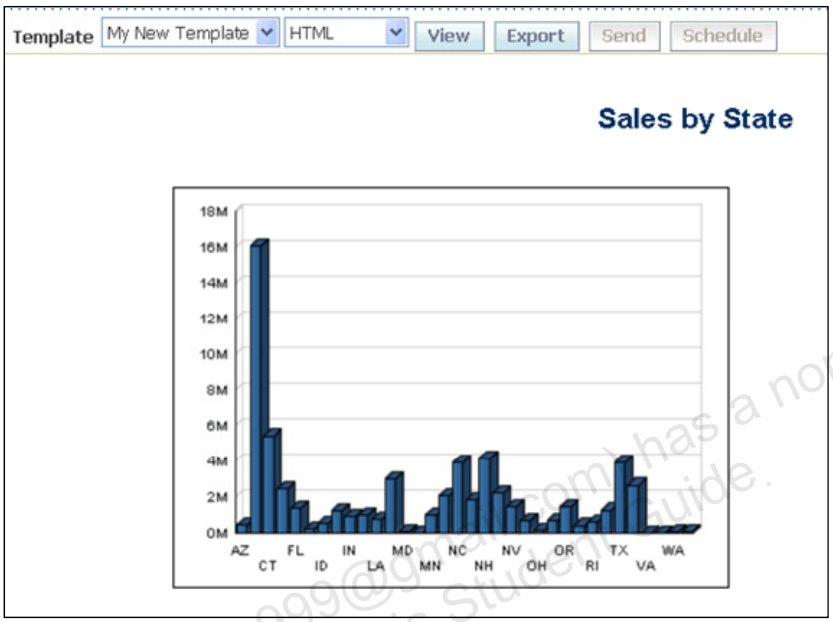

## 4. Uploading the Template

To upload the template, perform the following steps:

1. Select Oracle BI Publisher > Upload Template As.
2. In the "Upload as new" dialog box, specify a template name and click OK.

# Viewing the Report

Select the new report and click View.

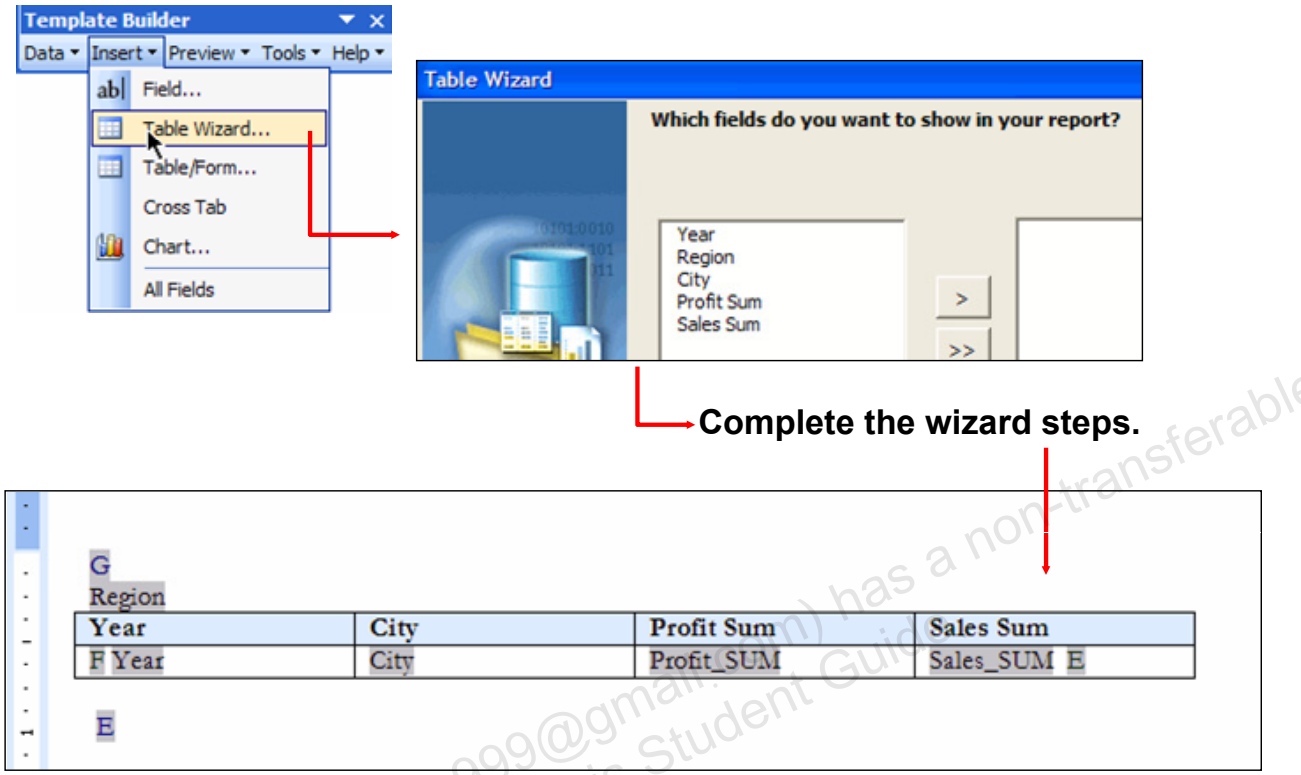





## 5. Viewing the Report

Before viewing the report, associate the uploaded template with the report by performing the steps described earlier in the lesson. To view the report, select it in BI Publisher Enterprise and click View.

# Creating a Template for the Report





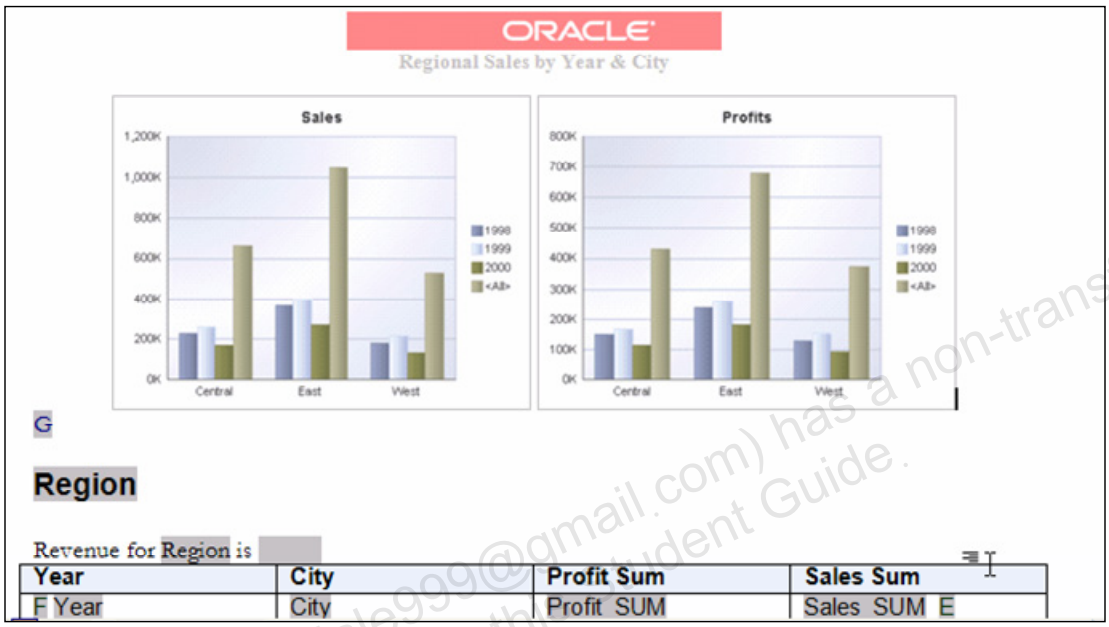

## 3a. Creating a Template for the Report

Use the Template Builder in BI Publisher Desktop to create a template for the Discoverer worksheet data. As discussed earlier in this lesson, you can apply any Word formatting and layout tools to the template to create a sophisticated presentation layout.

In the example, the Table Wizard is used to create a simple template for the Annual Sales, Profits by Region City worksheet that was previously selected as the Discoverer data source.

# Modifying the Template

Additional content items and MS Word formatting are added to the template.



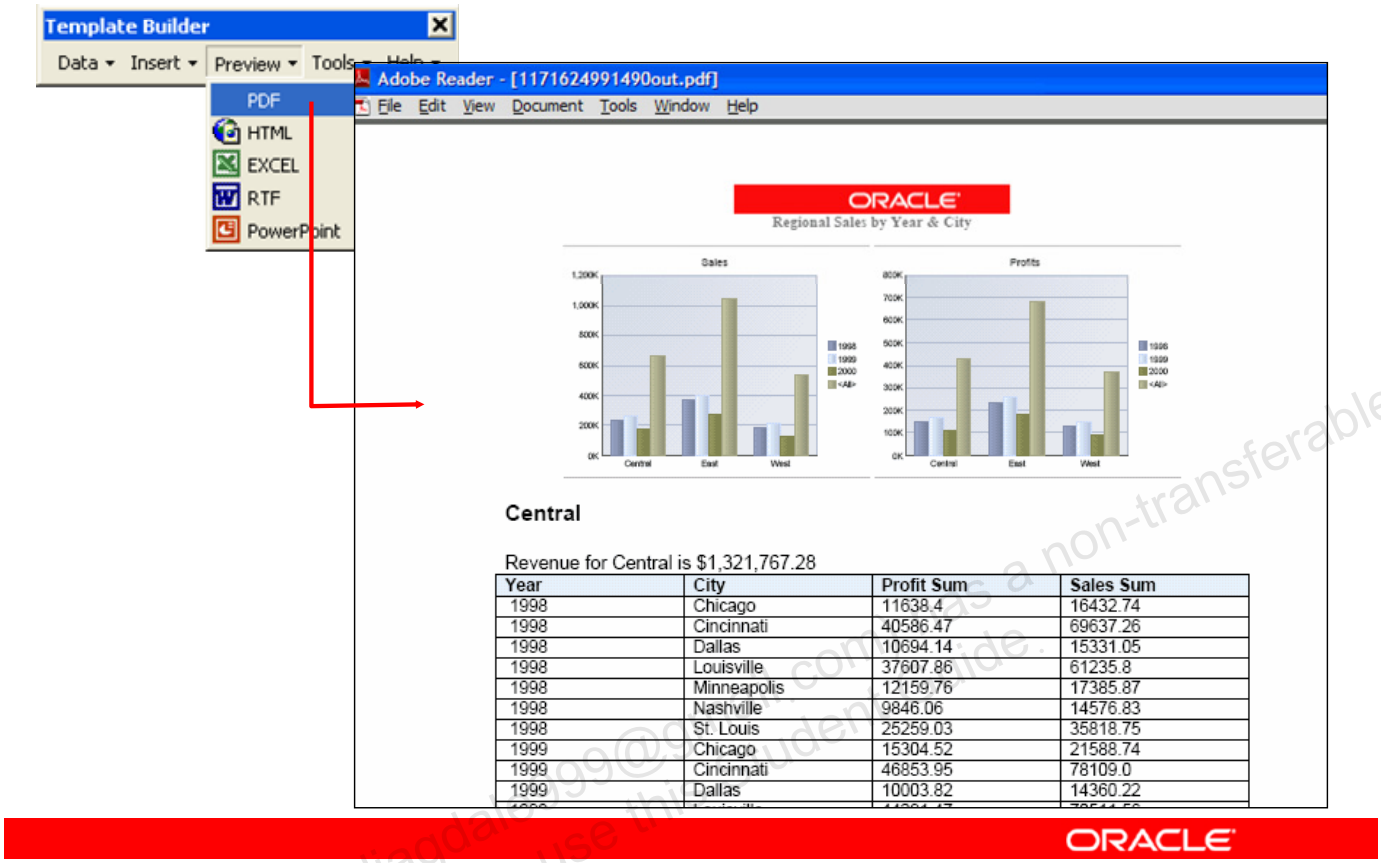



## 3b. Modifying the Template

In the example, the original template was modified in a number of ways:

- Word formatting was used to improve the presentation of the table headings and data.
- A data header was added to provide a sum of Revenue for the particular page item (Region).
- Two graphs were added: one displays Sales data, and the other displays Profits.
- A report header was added.

# Previewing the Report Output



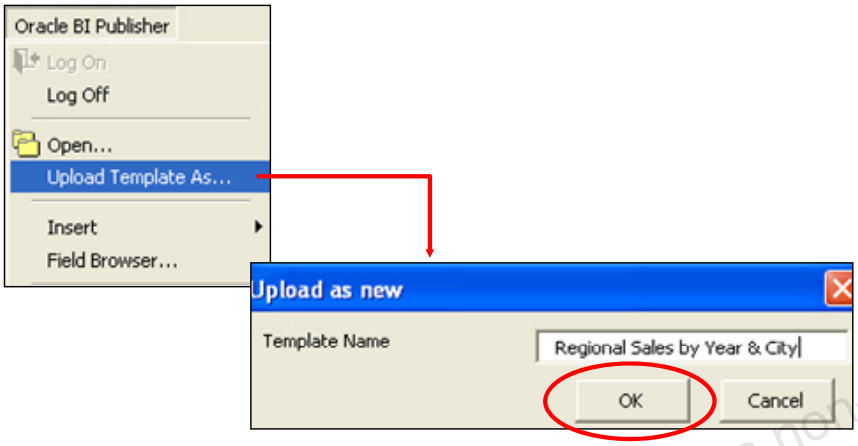



## 4. Previewing the Report Output

In the example, PDF format is chosen as the output format for the report. The report appears in the selected format.

The slide example shows a PDF preview of the report output by using the modified template.

# Uploading the Template






## 5. Uploading the Template

Upload the template to BI Publisher Enterprise as discussed earlier in this lesson.

# Summary

In this lesson, you should have learned how to:

- Create a BI Publisher report

# CASE LITE

***BI Publisher ……. Duration 0.5Day***



**Business Scenario:**

Vision Corporation Inc is a US based company, is in the field of manufacturing computer systems. It has sales offices across the globe and its sells to its customers through following options.

- Direct sales - For key customers with cumulative annual sales over 1 million USD per annum. They are treated as privilege customers, their early discount structure is decided by the top management and also the key customers have the first access to sample products for feedback before market launch.
- Distribution channels – For small account key customers, there are country specific re-seller. In this case the discount structure is fixed based on volume. The discount structure will be two folds
    - Reseller discount.
    - End customer discount – based on volume.



**Booking & Sales process:**

Vision Corporation receives Purchase order from end customer in case of direct sales and from its own distributor.

Vision Corporation is expected to maintain a sales system where customer purchase orders and sales invoices for customer are generated at Vision corporation central office. Once goods are ready for delivery invoices are generated.

Vision Corporation has its system assembly plants in Europe, US & Asia. The component sourcing operation for assembly of its product happens across the globe through centralized sourcing team based at US. The vision corporation also

maintains ware houses in US, Europe, Asia & Africa to store finished products so that it can cater to global market needs at shortest possible time.

The Top Management of the Vision Corporation needs to be periodically updated with certain reports in order to make business related strategic decisions for improving the Sales of the products which eventually increases the profits. As a developer, please create the following reports for submission to the top management.

1. Customer List
2. Product Sales Report

**Problem Statement:**

1. **Customer List**

Vision Corporation's sales manager requires all the active customer details to boost the sales by periodically informing the customers on the loyalty programs and promotions that are currently on. This list may be passed on to the Mailing department to mail the fliers and pamphlets to the customers. Mailing department requires the address details of the active customers in company letter head and also in PDF Format.

Refer Appendix B for a sample format. Please use the .rtf file provided for creating this report.

**Problem Statement:**

**2. Product Sales Report**

Sales manager of Vision Corporation requires a product wise report by customer. This report will enable the organization to make decisions on boosting the sale and also understand the buying patterns of a customer; this will also give vital clues to position the product better in the market.

Refer Appendix C for a sample format. Please use the .rtf file provided for creating this report.

**Bonus points:**

1. Exception handling in case of data not found.
2. Use conditional formatting to highlight a particular output with a different colour of your choice.

Vision Tech Corporation

# Customer Listing Report

Phone- 0331234555
techvision@ebmail.com

A-1, DB Road,
Mumbai , India

| Customer Name | Street | City | State | Country | Pin |
|---|---|---|---|---|---|
| | | | | | |

Vision Corporation

# Product Sales Report

Phone- 0331234555
techvision@ebmail.com

A-1, DB Road,
Mumbai , India
Pin-123432

| | | | |
|---|---|---|---|
| **Product Name** | | | |
| **Product Desc** | | | |
| **Customer Name** | | | |
| | **Quantity** | **Unit Price** | **Total** |
| | | | |
| **Total Quantity** | **Total Amount** | | |