Sub Templates in Oracle BI Publisher

An Oracle White Paper
Aug 2009

ORACLE

Sub Templates in Oracle BI Publisher

Sub Templates can be used to replace Sub Reports of Crystal Reports. Both non-linked and linked sub reports can be replaced by Sub Templates in BI Publisher. However, Crystal Reports generally creates a relationship between the main report and a sub report using related or unrelated data and such a relationship can be better handled in the Data Definition layer of BI Publisher to improve report performance.

OVERVIEW

Oracle BI Publisher supports several layout template types, e.g. RTF, PDF, XSL, excel, Flex, eText etc. The RTF template is the most common layout template type for a variety of document requirements. Sub template is a feature in RTF template that allows re-use of code, simplification of layout design and reduced maintenance. This feature is supported in E-Business Suite where you can upload sub template from Template Manager. On Enterprise Server, this feature can be used, however there is no user interface to upload and manage sub template. This whitepaper introduces the sub template feature and explains how to use it in the Enterprise Server.

INTRODUCTION

A sub template is a piece of formatting functionality that can be defined once and used multiple times within a single layout template or across multiple layout template files. This piece of formatting can be an RTF file format or an XSL file format. RTF sub templates are easy to design as you can use Microsoft Word native features. XSL sub templates can be used for complex layout and data requirements.

RTF SUB TEMPLATE

An RTF sub template consists of one or more <?template?> definitions. Following are the steps to apply RTF sub template to an RTF template.

1. Define RTF Sub Template

   Open an RTF file. Use the following syntax to create a template:

   ```
   <?template:templateName?>My Hello World !!<?end template?>
   ```

   If more templates are required, they all can be defined the same way in the same RTF file. Sequence of template definition in sub template file does not matter.

2. Call RTF Sub Template

The different <?template?> definitions in the sub template can be called into the RTF template. First step is to import the sub template file into the RTF template. You can use http or file protocol to import the file.

```
<?import:http://www.server.com/common/subTemplate.rtf
?>
```

```
<?import:file://C:/Files/common/subTemplate.rtf?>
```

Note:

1. Import statement is not required if sub template is defined in the RTF template file. In this case, the sub template definition is local to the RTF template and cannot be shared with other RTF templates.
2. EBS environment supports xdo protocol for sub template import. Example -

   <?import:xdo://OKC.OKCTCINC.en.00/?>

Next, call the <?template?> definitions in the RTF template using following syntax:

```
<?call:templateName?>
```

The position of the template is set by the call statement.

Similarly, call other <?template?> definitions in the sub template.

3. Set property to allow external file reference

   Go to report runtime configuration by clicking on 'Configure' link. Under properties tab, set "Disable External References" as false.

When to Use RTF Sub Template:

1. To re-use Common Layout

Often we have a situation when multiple reports have same header and footer content. Sub template is recommended in this case for header and footer so that in future if there is a change, it simply has to be done in one place and the change will reflect in all the reports. Follow the steps (1, 2, 3) under RTF sub template to re-use Common Layout.

2. To handle parameterized layout

If the requirement is to display a different layout based on a user parameter value or a List of Value selection, then the parameter can be passed to RTF layout and conditionally display the layout.

On the Report Editor you first need to define a parameter with name DeptName. If this parameter has to be associated with a List of Value (LOV), then create a LOV on the Report Editor page. Next, in the parameter definition select parameter type as "Menu" and then select the LOV from the selection.

In the RTF template define a parameter using syntax -

<?param@begin:DeptName?>

Now when user selects a department name from the List of Value on the report viewer page, the value gets passed to RTF layer into the DeptName parameter. To display the layout based on this user selection, you can use an IF statement or a CHOOSE statement to evaluate the parameter value and call the associated sub template. Use CHOOSE statement when too many conditional tests are to be done and a default action is expected for rest of the values, for example, for each department here we have a different sub template and if the user parameter has a department name with no associated sub template, then a default sub template can be called in otherwise section. For more information on CHOOSE statement, check the *BI Publisher Report Designer's Guide*.

```
<?choose:?>
  <?when:$DeptName='Accounting'?><?call:tAcc?>
  <?end when?>
  <?when:$DeptName='Sales'?><?call:tSales?>
  <?end when?>
  <?when:$DeptName='Marketing'?><?call:tMark?>
  <?end when?>
  <?otherwise:?><?call:tDefault?>
  <end otherwise>
<?end choose?>


Here all the templates in sub template - tAcc,
tSales, tMark and tDefault - are defined in one RTF
file.

<?template:tAcc?>
  ------ Accounting Layout ------
<?end template?>
<?template:tSales?>
  ------ Sales Layout -----------
<?end template?>
<?template:tMark?>
  ------ Marketing Layout -------
<?end template?>
```

```
<?template:tDefault?>
  ------ Default Layout ---------
<?end template?>
```

3. To handle dynamic / conditional layout

The sub templates can manage conditional layout based on report data, for example, address layout based on country code. It can be put in an IF condition statement, or can be a choose statement. A conditional display can be handled similar to the example in parameterized layout. We can also pass parameter to a <?template?> similar to how a parameter is passed to a function.

In this example address layout will be conditionally displayed based on the parameter passed to the <?template?>. Here we are passing two parameters – pCountry and pAddress.

```
<?template:tAddress?><?param:pCountry;string('USA')?>
<?param:pAddress;string('No Address')?>
   <?if:$pCountry='USA'?>
     USA Address layout: <?pAddress?>
   <?end if?>
   <?if:$pCountry="IN"?>
     IN Address layout: <?pAddress?>
   <?end if?>
<?end template?>
```

The <?template?> can now be called similar to other template call; however, to handle parameters we will have to enclose the parameters between <?call?> and <?end call?> tags. Here @inlines is a required context to include the content of this <?template?> seamlessly into the RTF template.

```
<?call@inlines:tAddress?>
  <?with-param:pCountry;./COUNTRY_CODE?>
  <?with-param:pAddress;./ADDRESS?>
<?end call?>
```

4. To handle simple calculations / repeating formulae

If your report has a lengthy calculation that is expected to be used at multiple places, then you can define such a calculation in a RTF sub template similar to a function.

```
<?template:calcInterest?>
  <?param:principal;0?>
  <?param:intRate;0?>
  <?param:years;0?>
  <?number($principal) * number($intRate) *
number($years)?>
<?end template?>
```

Note: when defining the parameter the default value is mandatory.

This template can be called from any place on the RTF template, even inside a for-each loop. Remember to keep the @inlines context.

```
<?call@inlines:calcInterest?>
  <?with-param:principal;./PRINCIPAL?>
  <?with-param:intRate;./INTEREST_RATE?>
  <?with-param:years;./NO_OF_YEARS?>
<?end call?>
```

.

Note: To handle complex and lengthy calculations /repeating formulae we recommend using XSL sub template instead of RTF sub template. The XSL sub template allows you to run the code in debug mode and can scale up better compared to RTF sub template.

XSL SUB TEMPLATE

XSL sub template empowers a report designer to handle complex data and layout requirements. You can transform the data structure for a section of report, e.g. charts, or you can create a style sheet to manage a complex layout. Similar to RTF template, XSL sub template consists of one or more <xsl:template> definitions.

1. Define XSL Sub Template

   XSL sub template consists of one or more XSL template definitions. These templates contain rules to apply when a specified node is matched.

```
<xsl:template
name="name"
match="pattern"
mode="mode"
priority="number">
  <!-- Content:(<xsl:param>*,template) -->
</xsl:template>
```

xsl:template	The **xsl:template** element is used to define a template that can be applied to a node to produce a desired output display
match=”pattern”	Optional. The match pattern for the template. **Note:** If this attribute is omitted there must be a name attribute
mode=”mode”	Optional. Specifies a mode for this template
name=”name”	Optional. Specifies a name for the template. **Note:** If this attribute is omitted there must be a match attribute
priority=”number”	Optional. A number which indicates the numeric priority of the template. It is quite possible that

	more than one template can be applied to a node. The highest **priority** value template is always chosen. The value ranges from -9.0 to 9.0

Example -

```
<xsl:template match="P|p">
  <fo:block white-space-collapse="false" padding-
bottom="3pt" linefeed-treatment="preserve">
     <xsl:apply-templates select="text()|*|@*"/>
  </fo:block>
</xsl:template>

<xsl:template match="STRONG|B|b">
  <fo:inline font-weight="bold">
     <xsl:apply-templates/>
  </fo:inline>
</xsl:template>
```

2. Import XSL Sub Template

   We have created import of XSL as separate step because an XSL sub template is always an external file and therefore import statement is mandatory.

   ```
   Using http protocol:
   <?import:http://www.server.com/common/subTemplate.xsl
   ?>

   Using file protocol:
   <?import:file:///C:/Files/common/subTemplate.xsl?>
   ```

3. Call XSL Sub Template

   The sub template call here is different from RTF template. Here templates defined in XSL sub template file are applied on data elements. There are two ways you can call a template defined in the XSL sub template.

   a. By matching the data content with match criteria:

      ```
      <xsl:apply-templates select="data_element"/>
      ```

      This will apply all the templates defined in the XSL sub template on the data_element and based on the data content appropriate function in those templates will be applied. This has been explained in the examples that follow in the next section under "to handle XHTML data" and "to dynamically apply formatting to a portion of data".

   b. By calling a template by name:

      ```
      <xsl:call-template name="templateName"/>
      ```

This will simply call the template by name and the template will execute similar to a function call. Here also parameters can be passed to the template call similar to RTF sub template.

```
<xsl:template name="templateName" match="/">
  <xsl:param name="name" />
</xsl:template>
```

This template can be called using syntax:

```
<xsl:call-template name=”templateName”>
  <xsl:with-param name="name" select="expression">
    <!-- Content:template -->
  </xsl:with-param>
</xsl:call-template>
```

This has been explained in the next section under “To transform data to handle complex chart requirements”

Note: XSL code syntax on RTF template should always be written in the BI Publisher Properties (or Text Form Field Options) dialog for a data placeholder. If the length of code exceeds the space in BI Properties dialog, you can split the code into multiple data placeholder.

4. Set property to allow external file reference

   On the Enterprise server, go to report runtime configuration by clicking on ‘Configure’ link. Under properties tab, set “Disable External References” as false. This will allow the report to reference an external file.

When to Use XSL Sub Template:

1. To handle XHTML data (i.e. XML data with HTML formatting)

Sometimes you have data stored with HTML formatting, and the requirement is to create a report using the HTML formatting available in the data. Following is a sample data with HTML formatting

```
<DATA>
  <ROW>
    <PROJECT_NAME>Project Management</PROJECT_NAME>
    <PROJECT_SCOPE>
      <p>Develop an application to produce
<i>executive-level summaries</i> and detailed project
reports. The application will allow users to: </p>
     <ol>
       <li>Import existing MS Project files </li>
       <li>Allow the user to map file-specific
resources to a central database entities (i.e.,
people) and projects; </li>
       <li>Provide structured output that can be
viewed by staff and executives. </li>
```

```
    </ol>
   </PROJECT_SCOPE>
   <PROJECT_DEFINITION><b>Information about current
projects is not readily available to executives.</b>
Providing this information creates a reporting burden
for IT staff, who may already maintain this
information in Microsoft Project files.
</PROJECT_DEFINITION>
  </ROW>
</DATA>
```

Note: The HTML must be in the XHTML format i.e.
 is not going to work it needs to be
</BR> for a new line. In other words all HTML tags should have start and end tags in the data.

XSL sub template comes as an easy solution to this requirement. Following is a template using XSL syntax to handle 'STRONG' or 'B' or 'b' in the XML data. The template then replaces the matched HTML string with its XSLFO equivalent.

```
<xsl:template match="STRONG|B|b">
 <fo:inline font-weight="bold">
  <xsl:apply-templates />
 </fo:inline>
</xsl:template>
```

Similarly you will have to write templates to handle different HTML formatting equivalents.

Next, this XSL sub template should be imported into the RTF template using syntax as shown below.

```
<?import:file:///c:/Files/sub/htmlmarkup.xsl?>
```

Or you can import using HTTP protocol as -

```
<?import:http://www.server.com/sub/htmlmarkup.xsl?>
```

The RTF template is now aware of the sub template, you now just need to call the functions therein to get the html converted. For each field in the RTF template that might have HTML markup you call for the markup functions to be applied. This requires a simple call:

```
<xsl:apply-templates select="PROJECT_SCOPE"/>
```

This command tells the processor to apply all templates to the value of the element PROJECT_SCOPE. It will then cycle through the sub template functions looking for a match. You need to do this for all the fields that may have HTML markup in the XML data.

Refer to the blog written by Tim Dexter for more details and sample files:
http://blogs.oracle.com/xmlpublisher/2007/01/02/

2. To dynamically apply formatting to a portion of data (e.g sub scripting / super scripting / chemical formula)

For documents that uses chemical formulae or a mathematical calculation, use of super script and sub script is a very common requirement. For example, in the sample XML data below CO2 is expected to be displayed as CO_2 and H2O is expected to be displayed as H_2O

```
<ROWSET>
 <ROW>
  <FORMULA>CO2</FORMULA>
 </ROW>
 <ROW>
  <FORMULA>H2O</FORMULA>
 </ROW>
</ROWSET>
```

This can be achieved by using XSL sub template. Using XSL syntax you can define a template with any name, say - "chemical_formatter" that will accept the FORMULA field as parameter, and then read one character at a time. It will compare the character with 0 – 9 digits and if there is a match then that character will be sub scripted using XSL FO syntax

```
<fo:inline baseline-shift="sub" font-size="75%">
```

Here is a sample code for the XSL template:

```
<xsl:template name="chemical_formatter">
<!-- accepts a parameter e.g. H2O -->
<xsl:param name="formula"/>
<!-- Takes the first character of the string and
tests it to see if it is a number between 0-9-->
<xsl:variable name="each_char"
select="substring($formula,1,1)"/>
<xsl:choose>
   <xsl:when test="$each_char='1' or $each_char='2'
or $each_char='3' or $each_char='4' or $each_char='5'
or $each_char='6' or $each_char='7' or $each_char='8'
or $each_char='9' or $each_char='0'">
<!-- if it is numeric it sets the FO subscripting
properties -->
     <fo:inline baseline-shift="sub" font-size="75%">
        <xsl:value-of select="$each_char"/>
     </fo:inline>
   </xsl:when>
   <xsl:otherwise>
   <!-- otherwise the charater is left as is -->
     <fo:inline baseline-shift="normal">
        <xsl:value-of select="$each_char"/>
     </fo:inline>
   </xsl:otherwise>
```

```
</xsl:choose>
<!--  test if there are other chars in the string, if
so the recall the template -->
<xsl:if test="substring-after($formula,$each_char)
!=''">
  <xsl:call-template name="chemical_formater">
    <xsl:with-param name="formula">
      <xsl:value-of
select="substring-after($formula,$each_char)"/>
    </xsl:with-param>
  </xsl:call-template>
</xsl:if>
</xsl:template>
```

Now, this XSL template has to be imported into the RTF template using syntax:

```
<?import:file:///C:/Files/sub/Chemical.xsl?>
```

Or you can import using HTTP protocol as –

```
<?import:http://www.server.com/sub/Chemical.xsl?>
```

Finally you need to call the sub template in the RTF template. Here you need to create a loop over the data and call XSL template by name :

```
<xsl:call-template name="chemical_formater">
<xsl:with-param name="formula"
select="FORMULA"/></xsl:call-template>
```

As this is a template call by name, we can as well use BI Publisher syntax, similar to RTF template:

```
<?call@inlines:chemical_formater?><?with-
param:formula;FORMULA?><?end call?>
```

This calls the formatting template with the FORMULA value e.g. H2O. Once rendered, the formulae will be shown as expected ie H_2O.

Refer to the blog written by Tim Dexter for additional details and related files: http://blogs.oracle.com/xmlpublisher/2006/04/09/

3. To transform data to handle complex chart requirements

Sometimes there are complex chart requirements that you may not be able to design using the Template Builder Chart Wizard. For example, if you have a requirement to display in a pie chart "Revenue from Top N Customers and Rest".

The data here has a flat hierarchy and is not in the order of Top N Customers and Rest format.

```
<ROWSET>
<ROW>
  <Products.Type>COATINGS</Products.Type>
  <Products.Brand>Enterprise</Products.Brand>
  <Markets.Region>CENTRAL REGION</Markets.Region>
  <Markets.District>CHICAGO
DISTRICT</Markets.District>
  <Periods.Year>2000</Periods.Year>
  <Measures.Dollars>1555548.0</Measures.Dollars>
</ROW>
  ------
  ------
</ROWSET>
```

Since BI Publisher inserts a dummy image for a chart, all the code for chart can be found under the Alt-Text tab of Format Image property for the chart image object. Say, if the requirement here is to display revenue from Top 5 Customers and one pie to show total revenue from rest of the customers, then first step will be to build a chart with Top 5 Customers. Later we can add the sixth pie for "Others". Following are the steps:

1) First we fix the number of data rows for BI Beans to 5 – instead of the complete data set.

```
chart:
<Graph depthAngle="50"
seriesEffect="SE_AUTO_GRADIENT" graphType="PIE">
<LegendArea visible="true" />
<LocalGridData colCount="1" rowCount="5">
```

2) We change the order of labels from Market District, created by grouping, to measure.Dollars per district by sorting.

```
<RowLabels>
<xsl:for-each-group select=".//ROW" group-
by="Markets.District">
<xsl:sort select="sum(current-
group()/Measures.Dollars)" data-type="number"
order="descending"/>
```

3) We need to select the first 5 elements – this is not mandatory for now but will be necessary for adding the rest of world. Note: XSL requires us to use the escape sequence < for "<".

```
<xsl:if test="position()&lt;6">
<Label>
   <xsl:value-of select="current-
group()/Markets.District" />
</Label>
</xsl:if>
```

4) We sort the data elements by revenue per district.

```
<xsl:for-each-group select=".//ROW" group-
by="Markets.District">
<xsl:sort select="sum(current-
group()/Measures.Dollars)" data-type="number"
order="descending"/>
```

5) We select only the first 5 elements – this is not mandatory for now but will necessary for the next step

```
<xsl:if test="position()&lt;6">
<RowData>
  <Cell>
    <xsl:value-of select="sum(current-
group()/Measures.Dollars)" />
   </Cell>
  </RowData>
</xsl:if>
```

6) Next we create the XSL sub template to calculate the revenue for "Others"

```
<xsl:template name="others">
  <xsl:variable name="list">
   <xsl:for-each-group select="/ROWSET/ROW" group-
by="Markets.District">
    <xsl:sort select="sum(current-
group()/Measures.Dollars)" order="descending"
data-type="number"/>
      <v><xsl:value-of select="sum(current-
group()/Measures.Dollars)"/></v>
   </xsl:for-each-group>
  </xsl:variable>
  <xsl:value-of
select="sum($list/v[position()&gt;5])"/>
</xsl:template>
```

This template groups all elements by "Market.District" and sort the resulting values by the sum of "Measure.Dollars" per District. The result is stored in the variable v.

Then we sum all elements in the variable with a position greater than 5 with the expression:

```
<xsl:value-of
select="sum($list/v[position()&gt;5])"/>
```

The escape sequence ">" means greater then or ">".

7) We need import this sub template from the RTF template. For testing we access the file from the file system – for the production implementation to store in on a web server and access it from there.

We add the following statement to the beginning of the document (not the chart) to import the sub template:

```
<?import:file:///C:/test/subtemplate.xsl"/>
```

Or use http protocol -

```
<?import:http://www.server.com/test/subtemplate.xsl?>
```

8) We increase the row Count from 5 to 6 – replacing step 1- for the additional "all others" slice.

```
<LocalGridData colCount="1" rowCount="6">
```

(9) We need to add a label for the summary of the remaining district

```
<RowLabels>
------------
<Label>ALL OTHER'S</Label>
</RowLabels>
```

(10) Finally we need to add another data row with the sum of all other districts by calling the function "others" defined in (6).

```
<RowData>
    <Cell>
     <xsl:call-template name="others"/>
    </Cell>
</RowData>
```

Note: Here we cannot use BI Publisher syntax <?call?> even though this is a template defined by name. This is because the entire chart code is in XSL syntax.

Refer to blog written by Klaus Fabian for additional details and related files:

http://blogs.oracle.com/xmlpublisher/2009/06/

4. To handle complex and lengthy calculations / repeating formulae

The XSL sub template is recommended to create an equivalent of a function, complex or lengthy calculation or formulae. The XSL sub template can be executed in debug mode, thereby allowing the designer to troubleshoot and resolve any issue. Moreover, the XSL sub template is scalable for large and complex data calculations.

CONCLUSION

Sub templates provide a great flexibility to handle some of the complex layout and data requirements. It is often used in different scenarios when migrating from any reporting tool that uses concepts of sub reports, functions, formulae etc. It allows report designer to separate out complex calculation into a separate file, thereby providing an organized and simplified layout design. Moreover, sub templates empower the RTF templates to extend functionality by use of XSL.

Sub Template in Oracle BI Publisher
Aug 2009
Author: [Author]
Contributing Authors: [Authors]

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com