Adding Data Elements to BI Publisher Documents

John Peters
JRPJR, Inc

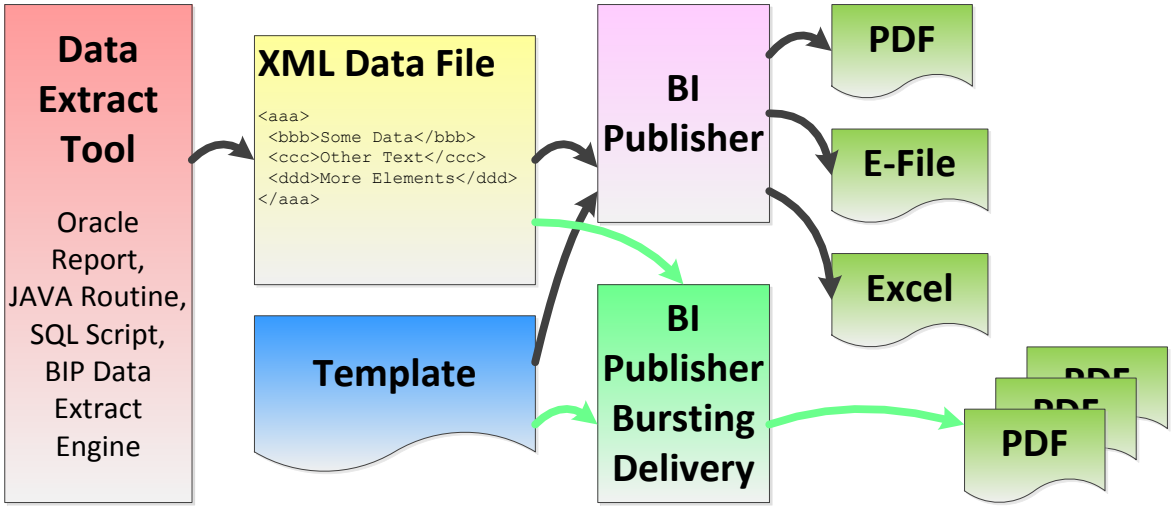
1 Abstract

This paper will cover the adding of data elements to BI Publisher documents. I will also cover a brief overview of BI Publisher and some general tips and recommendations.

This is revision 1 of my paper. Any updates to this presentation and paper will be posted on my web site: http:/jrpjr.com
Click on Paper Archives on the right hand side, then 'Collaborate12- Adding Data Elements to BI Publisher Documents'

2 Quick Introduction of BI Publisher

BI Publisher has been a great addition to the Oracle E-Business Suite. It enables us to create pixel-perfect external facing documents without the need for third party applications. The following diagram outlines the major components/processing steps associated with the generation of these documents. Each will be discussed in briefly below. There are many presentations on these topics in OAUG Conference Paper Database so I will not go into too much detail.



The process begins with the creation of an XML Data Stream using a Data Extract Tool. There are four commonly used tools used to accomplish this:
Oracle Reports (the predominate method even today in R12)
JAVA Routine (commonly used when documents are launched from an Oracle screen)
SQL Scripts
BI Publisher Data Extract Engine

The output from the Data Extract Tool is a simple XML Data File. You can open these files in your web browser to review the structure and contents of the data file. When running an Oracle Report as a concurrent program you can view the XML Data File by clicking on:

View Requests => Diagnostics => View XML

You create a Layout Template using a variety of tools. The most common is Microsoft Word using the BI Publisher add-in (saving the file as an RTF file). The Layout Template basically has fields that correspond to tags in the XML Data File. It can also contain logic to control the display of data as well as simple arithmetic operations.

BI Publisher then combines the XML Data File and Template to generate the output document which can be in many formations. The most common formats are PDF, E-File and Excel. If the XML Data File contained many transactions worth of data, as an example, many Customer AR Invoices then the resulting single PDF will contain all Customer AR Invoices from this run. This rarely does a business any good, so BI Publisher has a Bursting and Delivery component that once again combines the XML Data File and Template to generate individual documents based on rules you configure the bursting control file with. There is also a Delivery component that can:

Email Files
Fax Files
FTP Files

As well as many other user extensible possibilities.

Again please refer to the OAUG Conference Paper Database for more details on these topics.

3 General Recommendations and Tips

The following are some general recommendations and tips on the overall BI Publisher process described above.

3.1 XML Data

I would suggest that you prefix your new custom XLM tags (when you add data elements) with Oracle's recommended XXCUST naming convention. This follows Oracle's general customization standards. It makes it easier to find customizations you have done when looking at these routines in the future. And lastly your custom tags won't be duplicated by Oracle's seeded tags.

3.2 Layout Template

As I described above when Bursting/Delivering documents BI Publisher merges the XML Data File and Layout Template a 2nd time. This can actually be data dependent so you could call different templates based on the state of data in each XML record block. If is very important in the Bursting Control file to pull the Layout Templates from the database, not the file system. Most of Oracle E-Business Suite examples and documents describe only how to pull the Layout Templates from the file system. The following is how you pull the Layout Templates from the database:


```
<?xml version="1.0" encoding="UTF-8"?>
 <xapi:requestset xmlns:xapi="http://xmlns.oracle.com/oxp/xapi" listener="oracle.apps.xdo.batch.SampleListener">
  <xapi:request select="/XXCUST_RAXINV/LIST_G_ORDER_BY/G_ORDER_BY/LIST_G_INVOICE/G_INVOICE">
   <xapi:delivery>
    <xapi:filesystem id="ARI" output="/interface/${DB_NAME}/DOCUMENTS/IN/ARI_${CUSTOMER_NUMBER}_${TRX_NUMBER}.pdf">
    </xapi:filesystem>
   </xapi:delivery>
   <xapi:document key="${TRX_NUMBER}" output-type="pdf" delivery="ARI">
    <xapi:template type="RTF" location="xdo://XXCUST.XXCUST_RAXINV.en.US/?getSource=true" translation="" filter="">
    </xapi:template>
   </xapi:document>
  </xapi:request>
 </xapi:requestset>
```

In the example above the key line in the bursting control file is the one highlighted in yellow with a red font. The location specifies that we are to pull the template from the database. The following are the elements of this string:
<app_code>.<template_code>.<language_code>.<country_code>

Where in the example above:
<app_code> = XXCUST
<template_code> = XXCUST_RAXINV
<language_code> = en
<country_code> = US

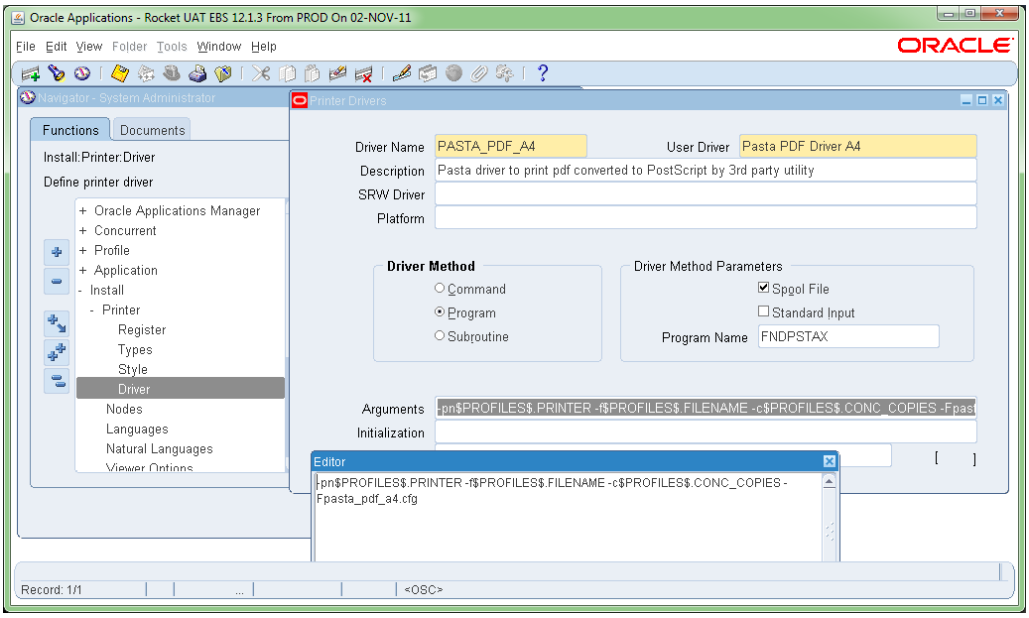
These values correspond to the values you put in on the BI Publisher Template Registration page. It is very important to pay particular attention to the Language and Country since they are optional on the screen and once you save the Layout Template you cannot change these values.

3.3 Handling Letter and A4 Paper Sizes

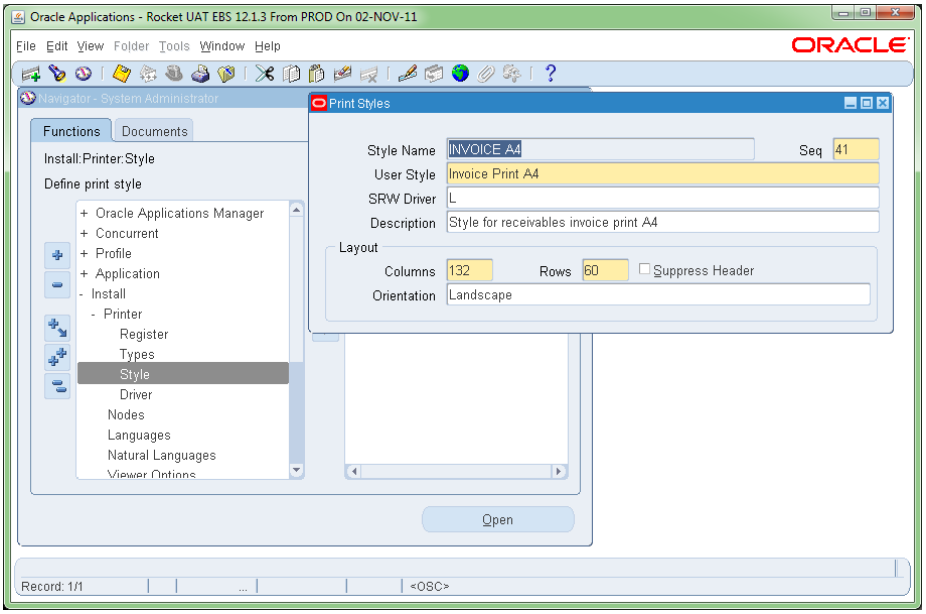
Don't create separate Layout Templates for Letter and A4 paper sizes. Scale your output using PASTA printing. This is how you do this:

1) Create Printer Driver PASTA_PDF_A4

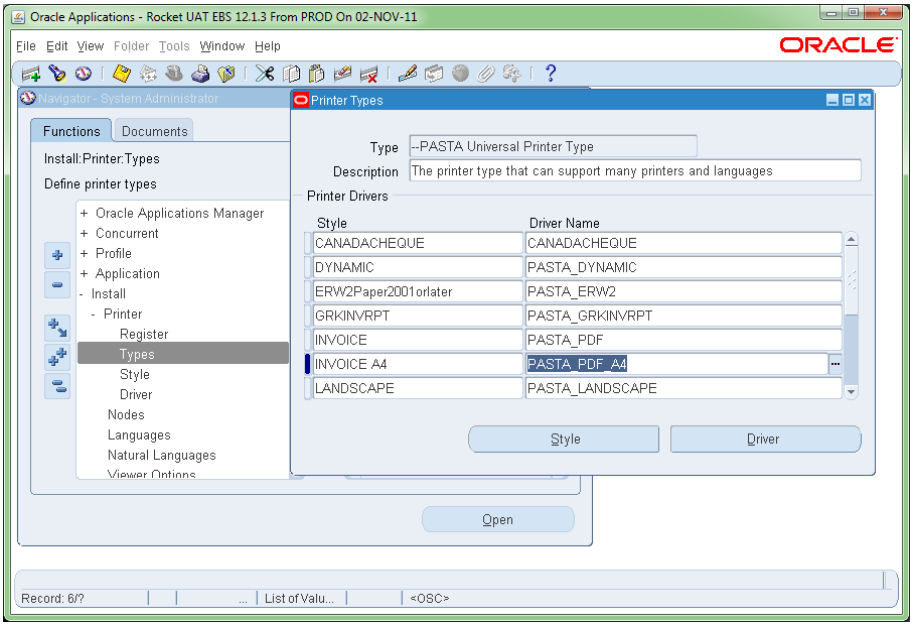
Arguments line:
-pn$PROFILES$.PRINTER -f$PROFILES$.FILENAME -c$PROFILES$.CONC_COPIES -Fpasta_pdf_a4.cfg



2) Create Print Style (something like INVOICE A4)



3) Add Print Style and Print Driver to Printer Type

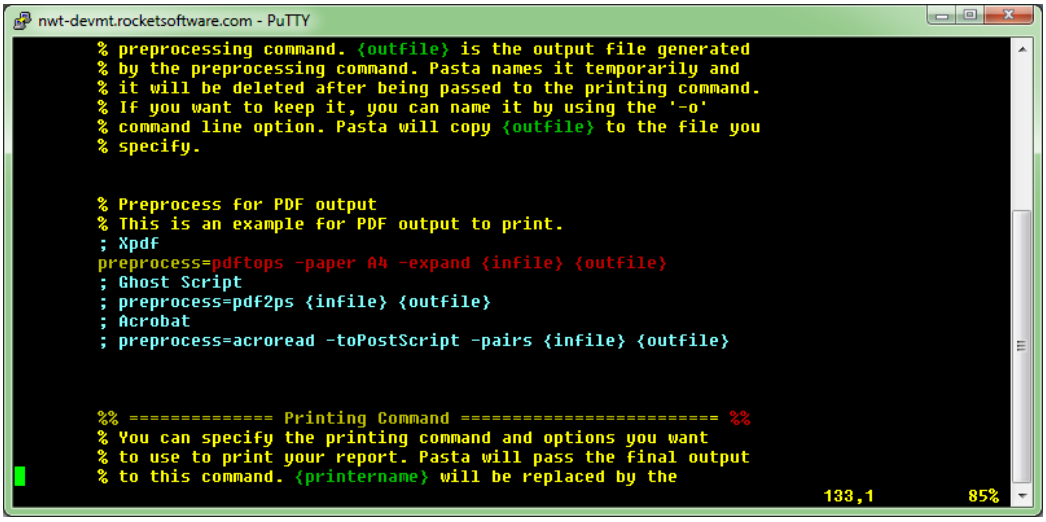


4) Create Pasta Config File pasta_pdf_a4.cfg

cp $FND_TOP/resource/pasta_pdf.cfg $FND_TOP/resource/pasta_pdf_a4.cfg

Edit following line:

preprocess=pdftops -paper A4 -expand {infile} {outfile}



This example uses the conversion utility pdftops (PDF to PostScript). There are equivelants for the other two conversion utilities.

BI Publisher makes it fairly easy to get a nicely formatted output from already provided XML Data Streams. However, you often need to add data element to these XML Data Streams. This is what this presentation is all about.

4 Common Data Element Additions

The following are common data element additions I have had to make a many client sites.

4.1 User Data Elements

The following are common user data elements that I need to add:

Descriptive Flex Fields – they were important enough to collect and often they affect output

Legal Entity Address Data – if is often not enough to print a logo on a document often in multi-organization implementations a legal entity name and address block must be printed on the document. These could be hard coded in templates, but is much cleaner to code these as data extracts from the database so changes and additions can be easily made by the users.

Data from a foreign keyed record – You are printing an AR Invoice and you want something that was stored on the customer record. Or the AR Invoice must reference something from the Sales Order.

The most important thing to remember when these requests come up is you should always ask the following first before adding the data element:

"Isn't this a common data requirement? Why are we different?"

I will typically find that the missing data element is just a missed setup or configuration that was not known or required until now. Do some digging before you extend!

4.2 For Bursting/Delivery

The following are common user data elements that I need to add to support Bursting and Delivery:

Instance Name – You don't want to burst/deliver the same in PROD and non-PROD instances. BI Publisher Delivery Manager does not use Oracle Workflow to send emails. It communicates directly to an SMTP Server. So there is no "Test Address" functionality, you must design for it yourself.

Date/Time Stamp for filenames – Many times I need a YYYMMDDHHMMSS timestamp to put into an output filename. It is best to generate this timestamp at the exact moment the data is extracted so it is consistent.

Delivery Details – Data elements like Email Address, Userid/Passwords for file transfers, etc. Oracle can't handle all of your cases you will probably need to add these yourself.

4.3 Language Requirements

Oracle out of the box supports multiple languages. But most of my clients don't implement full blown MLS, with different forms and reports. They just need a few external facing documents to include some regional language titles. You could create different templates, one for each language. You can use sub-templates, but this tends to generally get complicated.

Instead pull the template boiler plate text from a custom table or a %_TL table. Take a look at ML Note 1077709.1: 'Using Lightweight MLS With Oracle E-Business Suite Release 12.1.3'. This is a new feature

that allows you to have access to the %_TL tables to put in language translations without the overhead of the MLS Forms and Reports. My clients often just need the boiler plate text translated. For this I have used the following:

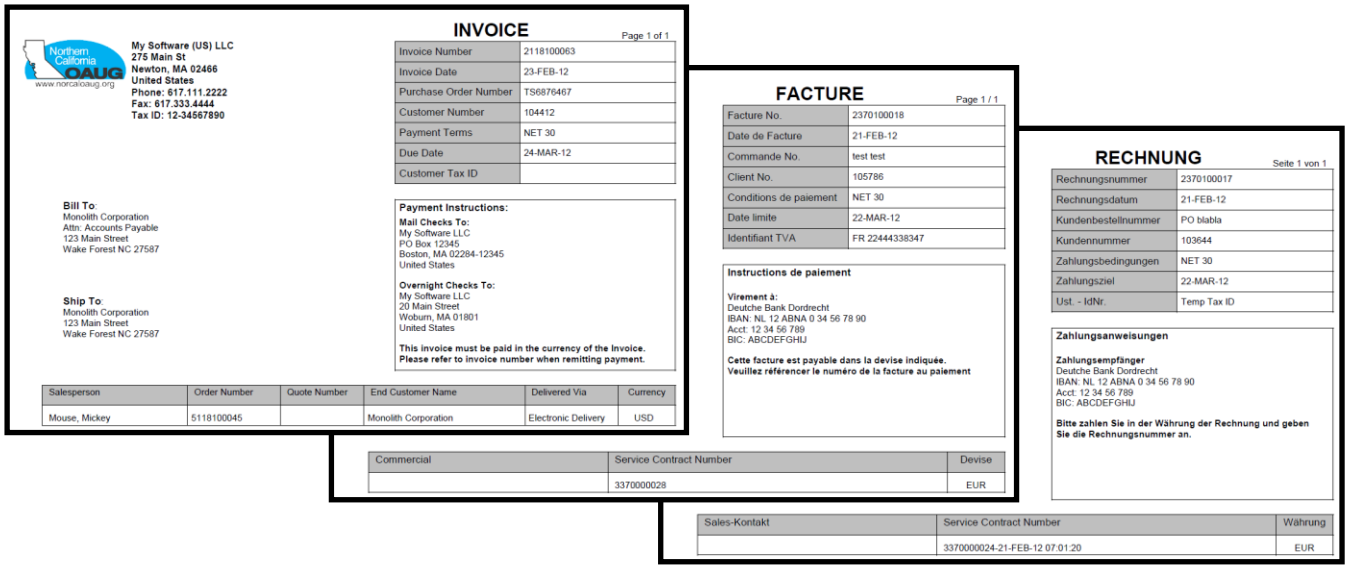
Custom PL/SQL Function

```
FUNCTION GET_MEANING (p_lookup_code     varchar2,
                      p_language_code   varchar2 default 'US',
                      p_org_id          number default 0
                     )
  return varchar2;
```

Pulls from Custom Table

LOOKUP_CODE	LANGUAGE_CODE	ORG_ID	LOOKUP_MEANING
LBL_TITLE_INVOICE	DE	0	RECHNUNG
LBL_TITLE_INVOICE	FR	0	FACTURE
LBL_TITLE_INVOICE	US	0	INVOICE

This allows me to have one template that supports many different languages for the boiler plate text.



So we have seen how important it is to add Data Elements to the XML Data Extract. Now how do we add these data elements.

5 Capabilities Provided by Oracle to Add Data Elements

Oracle development has provided some hooks we can use to add additional data elements. Some are supported some are not. The two that I am most familiar with are centered around: R12 Payments, R12 PO Documents, each will be discussed in more detail in the following sections.

Are there others? These two are just the ones I know about. Please send me an email if you know of other hooks provided by Oracle Development so we can share them with rest of the user community.

5.1 R12 Payments

If you want to have DFF's displayed in the payments XML data it is often as easy as registering them to be displayed in the EBS. Once a DFF is setup and displayed it should often come out in the XML Data File. No customizations are necessary.

For other data elements there is a stub PL/SQL package which will be run adding data to the XML Data File. Take a look at ML Note: 457539.1 for details on how to add your own queries into the PL/SQL package IBY_FD_EXTRACT_EXT_PUB.

You can add data to the following level of the XML Data File:

XML Data File Level	PL/SQL Function	Parameter
Instruction	Get_Ins_Ext_Agg	p_payment_instruction_id
Payment	Get_Pmt_Ext_Agg	p_payment_id
Document Payable	Get_Doc_Ext_Agg	p_document_payable_id
Document Payable Line	Get_Docline_Ext_Agg	p_document_payable_id
Payment Process Request	Get_Ppr_Ext_Agg	p_payment_service_request_id

Create a custom cursor that uses the parameter from the corresponding outer function call in the table above. Changes to the package body IBY_FD_EXTRACT_EXT_PUB may have to be reported after patching.

Here is an example of adding two data elements to the Payment level of the XML Data File.

```
FUNCTION get_pmt_ext_agg (p_payment_id IN NUMBER)
  RETURN XMLTYPE
IS
  l_pmt_ext_agg   XMLTYPE;
  CURSOR l_pmt_ext_csr (p_payment_id IN NUMBER)
    IS
    SELECT XMLCONCAT (XMLELEMENT ("XXCUST_EXT_AGG",
                                  XMLELEMENT ("XXCUST_FileID", cba.attribute1),
                                  XMLELEMENT ("XXCUST_CompanyID", cba.attribute2)
                                 )
                      )
      FROM IBY_PAYMENTS_ALL ipa,
           CE_BANK_ACCOUNTS cba
     WHERE ipa.internal_bank_account_id = cba.bank_account_id
       AND ipa.payment_id = p_payment_id;
BEGIN
  OPEN l_pmt_ext_csr (p_payment_id);
    FETCH l_pmt_ext_csr
     INTO l_pmt_ext_agg;
  CLOSE l_pmt_ext_csr;
  RETURN l_pmt_ext_agg;
END get_pmt_ext_agg;
```


The block of code highlighted in yellow is the cursor that pulls the two new data elements from the database. The block of code highlighted in blue is where the cursor is opened and the XML data is placed into an XML data variable and returned from the function. Oracle then appends this XML Data to the data it has generated internally. Also notice that we use functions XMLCONCAT and XMLELEMENT to correctly format the results as an XML data stream.

5.2 R12 PO Documents

The following is not supported by Oracle, but I have used in the past and there are several references on OTN of others doing the same. The JAVA program that builds the Printed PO is difficult to modify. You can't get access to the source code, and using a JAVA de-compiler produces almost worthless output.

However, the following views are the basis for the output in the XML Data Stream:
PO_HEADERS_XML
PO_LINES_XML
While this is not supported by Oracle, I have safely modified these views to include new data elements.

I have also seen reference that others have changed the following views:
PO_LINE_LOCATIONS_XML
PO_DISTRIBUTION_XML
PO_RELEASE_XML
Of course changes to any of these views may need to be re-ported after patching. See the next chapter for ways to minimize your re-porting effort. You have to keep your eyes on the versions of the ODF files that generate them to know when you need to put your fixes back in.

6 Modifying Oracle's Reports

This is probably the most common method I have seen or used for adding XML Data Elements. Quite often you only need one or two data elements, they are from the same tables already involved in the select statements and it is just easier to add those columns to the select statement. Whenever modifying Oracle's RDF's you should always:

1) Copy the RDF to your custom top directory
2) Copy the concurrent program definition to your custom application

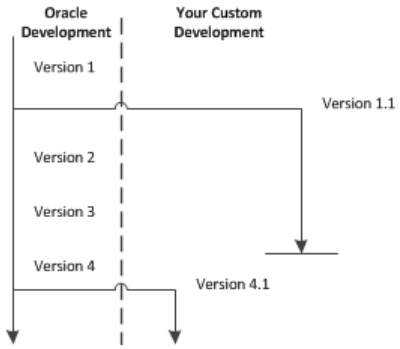
If the program is one that must exist in the seeded Oracle Application (like some Payables ones) I do the following:

1) Copy the RDF to your custom top directory
2) Copy the concurrent program definition keeping it in the same application
3) Rename the original RDF with something like '_BAK' or '_YYYYMMDD' suffixed to the filename
4) Place a relative symbolic link for your custom RDF from your custom top directory to the application top directory, there by replacing the original filename.

```
cd $XXCUST_TOP/reports/US/
ln -s ../../../../fnd/12.0.0/reports/US/ORACLES.rdf MY_VER_ORACLES.rdf
```


Some drawbacks to this approach of adding Data Elements (actually of any modifications you make to Oracle's RDF):

1) You have branched Oracle's development.



2) You are responsible for porting all future patches, upgrades to your code tree. Often you can skip some versions, but this depends upon the expanse of your implementation and your patching.

3) You need an Oracle Reports Developer

Here are some steps you can take to minimize these drawbacks:
1) Always copy Oracle's code
2) Keep track of Oracle's changes, verify if you need to uptake them
3) If you are just adding something from the same table, just add the column in the select statement
4) I would strongly suggest that if you are joining to other tables not already in the query to add them as PL/SQL functions. This makes it much easier to report the changes in the future.
5) You will need an Oracle Reports Developer. Either hire one, train one internally, or align yourself with an outside firm.

7 Modify by Porting to Oracle's BI Publisher XML Data Extract Engine

This is the only option you have with Oracle's JAVA Concurrent Programs. You can't get access to the source code. JAVA de-compilers produce code that is difficult to work with. An example of this is the Service Contract Quote, which can be run as a Concurrent Program or called from within the Oracle Service Contracts Launchpad. Here are the steps you should follow if you want to take this approach.

The drawbacks with this approach are the same as modifying Oracle's RDF, except you can sometimes simplify the query for your specific case and the BI Publisher Extract Engine is a little simpler to use than Oracle Reports. In addition, the BI Publisher XML Data Extract Engine is faster and more efficient than Oracle Reports.

7.1 Write the SQL Query

You don't have to start the SQL Query from scratch. Get Oracle's primary cursors from the Concurrent Program. You often don't need the full set of queries, most of the time these queries are overly complex because they have to handle all of the possible business scenarios and Oracle modules involved. Your site is probably far simpler, just take what you need. There are a couple of options.

1) Open an RDF and extract what you need from it.

2) Trace the Concurrent Program, dig through TKPROF output. This requires both functional and technical knowledge.

There is another option with RDF's. Oracle BI Publisher Release 10.1.3.3 has a conversion utility to take an Oracle Report and convert it to a BI Publisher report. The queries are converted to a PL/SQL Package. I have not used this but it is available as an option.
Oracle Business Intelligence New Features Guide, Release 10.1.3.4.2
Part Number E10416-07

7.2 Put Query into XML Data Template

This will use the BI Publisher Data Extract Engine, not to be confused with the BI Publisher Layout Template. The XML Data Templates have the query embedded in a XML formatted file. The XML Data Template is read by the BI Publisher Data Extract Engine and processed. With XML Data Templates you have similar functions to Oracle Reports:
- Multiple Queries and Joins
- Pre/Post Event Triggers
- Even Distributed Queries that cross multiple databases

The following is the basic structure of the XML Data Template file.

```
Data Template
   Properties
   Parameters
   Lexicals
```

Data Query
Data Structure

Here is an example of an XML Data Template file with each section of the structure identified above highlighted. Some structures like the Parameters and Lexicals can be left out using the XML tag structure shown below.

```
<dataTemplate name="HELLOWORLD" defaultPackage="" description="Hello World DT">
  <properties>
    <property name="include_parameters" value="true"/>
    <property name="include_null_Element" value="true"/>
    <property name="xml_tag_case" value="upper"/>
    <property name="db_fetch_size" value="500"/>
    <property name="scalable_mode" value="off"/>
    <property name="include_rowsettag" value="false"/>
    <property name="debug_mode" value="off"/>
  </properties>
  <parameters/>
  <lexicals/>
  <dataQuery>
    <sqlStatement name="Q1" dataSourceRef="">
      <![CDATA[
        select 'Hello World!' WELCOME from dual
      ]]>
    </sqlStatement>
  </dataQuery>
  <dataStructure>
    <group name="HELLO" source="Q1">
     <element name="WELCOME" value="WELCOME"/>
  </group>
  </dataStructure>
</dataTemplate>
```

7.3 Layout Template

Just like other BI Publisher report outputs you will need to create the Layout Template. This can be done using Microsoft Word and he BI Publisher Add In. This is beyond the scope of this paper, please refer to the OAUG Conference Paper database for many presentations on this topic.

7.4 Pre Event Triggers

When creating the XML Data Template you can include Pre Event Triggers that are executed before your queries are run. These are especially helpful if you are running in BI Publisher Standalone version and connecting to your Oracle EBS database, when you need to initialize your Multi-Org session environment.

If you are running the Oracle EBS version of BI Publisher you will be using the executable XDODTEXE to run your XML Data Templates. This program's framework already initializes your Multi-Org session environment so you don't need to do this.

7.5 Post Event Triggers

Post Event Triggers are essential if you want to call the BI Publisher Bursting Engine. This will allow you to run your Oracle EBS concurrent program calling the XML Data Extract Template. This will also automatically submit the Bursting Engine Concurrent Program to burst and deliver your output files. There are several steps to accomplish this:

1) You need to register a default package on your dataTemplate line of the

```
<dataTemplate name="XXCUST_OKSDATA"
    description="XXCUST Service Contract Quote"
    defaultPackage="XXCUST_OKS_QUOTE_CP" version="1.0"
 >
```

2) Between the </group> and </dataStructure> tags you need to put in a new data element that is a count of the number of records you received from your query. This is important because you don't want to call the Bursting program if no output was generated. It will error in this case.

```
  </group>
  <element name="OKS_COUNT" function="count" dataType="number" value="G_HEADER.H_Contract_ID"/>
  </dataStructure>
```

3) Between the </dataStructure> and </dataTemplate> tabs you need to put in a call to a function within your default package that will call FND_REQUEST.SUBMIT_REQUEST. You need to pass the count parameter from the prior step.

```
 <dataTrigger name="afterReportTrigger"
    source="XXCUST_OKS_QUOTE_CP.afterreport(:OKS_COUNT)"
  />
```

4) Your PL/SQL Function will look something like the following

CREATE OR REPLACE package body APPS.XXRKT_OKS_QUOTE_CP as

```
FUNCTION AFTERREPORT (p_oks_count number)
  return boolean
IS
  v_request_id number;
BEGIN

  if (nvl(p_oks_count,0) > 0)
  then
    v_request_id := FND_REQUEST.SUBMIT_REQUEST ('XDO',  -- application
                                                'XDOBURSTREP',  -- program
                                                '',  -- description
                                                '',  -- start_time
                                                FALSE,  -- sub_request
                                                'Y',  -- argument1
                                                FND_GLOBAL.CONC_REQUEST_ID,  -- argument2
                                                'Y',  -- argument3
                                                chr(0),'', '', '', '', '', '',
                                                '', '', '', '', '', '', '', '', '', '',
                                                '', '', '', '', '', '', '', '', '', '',
                                                '', '', '', '', '', '', '', '', '', '',
                                                '', '', '', '', '', '', '', '', '', '',
                                                '', '', '', '', '', '', '', '', '', '',
                                                '', '', '', '', '', '', '', '', '', '',
                                                '', '', '', '', '', '', '', '', '', '',
```

```
                                                  '', '', '', '', '', '', '', '', '', ''
                                                );
  end if;
  return (TRUE);
END;

END XXRKT_OKS_QUOTE_CP;
/
```

8 Conclusion

BI Publisher is a very powerful tool for generating quality looking output files. Often you will need to go to add to Data Elements to the XML Data File that is generated. This paper showed a variety of ways to accomplish this. For recent copies of this paper/presentation please download the most recent presentation from my web site:
http://jrpjr.com/
Click on Paper Archives on the right hand side,
then Collaborate12- Adding Data Elements to BI Publisher Documents

