Oracle® Transportation Management

Report Designer's Guide

Release 6.3

Part No. E38433-01

November 2012

ORACLE

Oracle Transportation Management Report Designer's Guide, Release 6.3

Part No. E38433-01

Contents

Send Us Your Comments

Oracle Transportation Management Report Designer's Guide, Release 6.3

Part No. E38433-01

Oracle welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the title and part number of the documentation and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: otm-doc_us@oracle.com

If you would like a reply, please give your name, address, telephone number, and electronic mail address (optional).

If you have problems with the software, contact Support at https://support.oracle.com or find the Support phone number for your region at http://www.oracle.com/support/contact.html.

Preface

This document is intended for Oracle Transportation Management clients, Oracle Transportation Management System administrators, or Oracle Transportation Management Consultants who have an interest in creating or customizing reports intended for use within the Oracle Transportation Management Application.

Change History

Date	Document Revision	Summary of Changes
11/2012	-01	Initial release Added the Using a Saved Query as a Report Input Parameter section. Added information on supporting CSV format for report layouts. Added information on supporting XSL-FO stylesheets for eText output.

1. Reporting Overview

This document describes how to design and generate reports for Oracle Transportation Management. It is intended for Report designers and integrators who need to develop reports based off of Oracle Transportation Management data, and for system administrators who need to configure and tune report generation. It is not intended to serve as an installation or users guide. Please consult the Installation Guide for details on installing Oracle Transportation Management; the online help for details on generating ad-hoc or scheduled reports from within Oracle Transportation Management.

Architecture

Oracle Transportation Management supports two methodologies to generate reports:

- **External**: Report requests are sent via HTTP to an external report server. The report server may begin an interactive session with you or simply return report content.
- **Embedded**: Report requests are routed to an Oracle Transportation Management application server. The server uses Oracle's Business Intelligence Publisher (BI Publisher) libraries to generate report content.

Figure 1 summarizes the possible data flows.

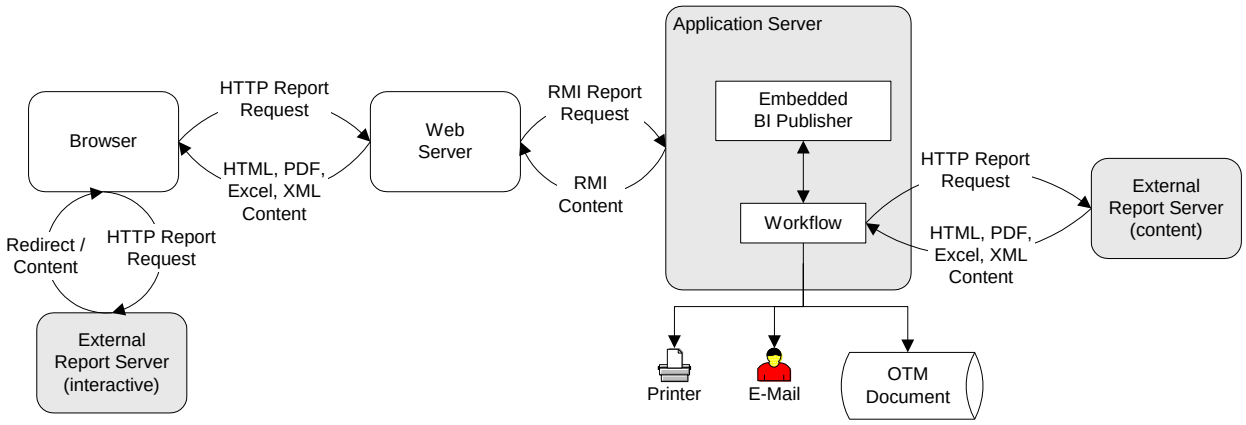


Figure 1: Reporting Architecture

External Generation

When providing an external report link to the browser, Oracle Transportation Management checks the type of external report.

If the external reporting system does not handle embedded content, the link directly maps to the report system URL. After selecting the report, you are redirected to the external report system. This system may prompt for parameters and/or distribution, generate the report, and return report content to the browser. Oracle Transportation Management makes no assumptions regarding any returned content.

If the external reporting system, though, handles embedded content, the report link connects to standard Oracle Transportation Management reporting screens. These screens allow you to run ad-hoc or scheduled reports. For ad-hoc reports, an HTTP request is sent to a web server which passes the request via RMI to an application server. The application server sends an HTTP request to the report server and assumes the response, if successful, contains report content. This content is piped back

through RMI to the web server, and then through HTTP to the browser. The content may also be distributed via e-mail, IPP printing, and/or stored with an associated business object.

Scheduled report requests, or requests triggered by a workflow agent, similarly send an HTTP request to the report server. The response content is distributed via e-mail, printed via IPP, or stored with an associated business object.

Embedded Generation

With embedded generation, the report link connects to standard Oracle Transportation Management reporting screens. These screens allow you to run ad-hoc or scheduled reports. For ad-hoc reports, an HTTP request is sent to a web server which passes the request via RMI to an application server. The application server invokes BI Publisher APIs to generate the report within the server's JVM[1]. The resulting content is piped through the web server back to the browser. The content may also be distributed via e-mail, IPP printing and/or stored with an associated business object.

Scheduled report requests, or requests triggered by a workflow agent, similarly invoke BI Publisher from the application server. The response content is distributed via e-mail, printed via IPP, or stored with an associated business object.

BI Publisher Concepts

Oracle Transportation Management is tightly integrated with BI Publisher to deliver high quality, multiple format output, translation compliant, and easily customizable reports which span the entire application. The advantages of BI Publisher include the reduced cost of report maintenance, a consistent user interface, the ability to generate reports in multiple output formats, and the ability to use already familiar tools to develop report layouts. BI Publisher enables the separation of the report data from the report design layout, as well as, report translations. It provides flexibility in the creation, modification, and maintenance of reports.

Figure 2 summarizes the process flow for BI Publisher report generation.

[1] See section 0 for properties to control which tier, web or application, generates report components.

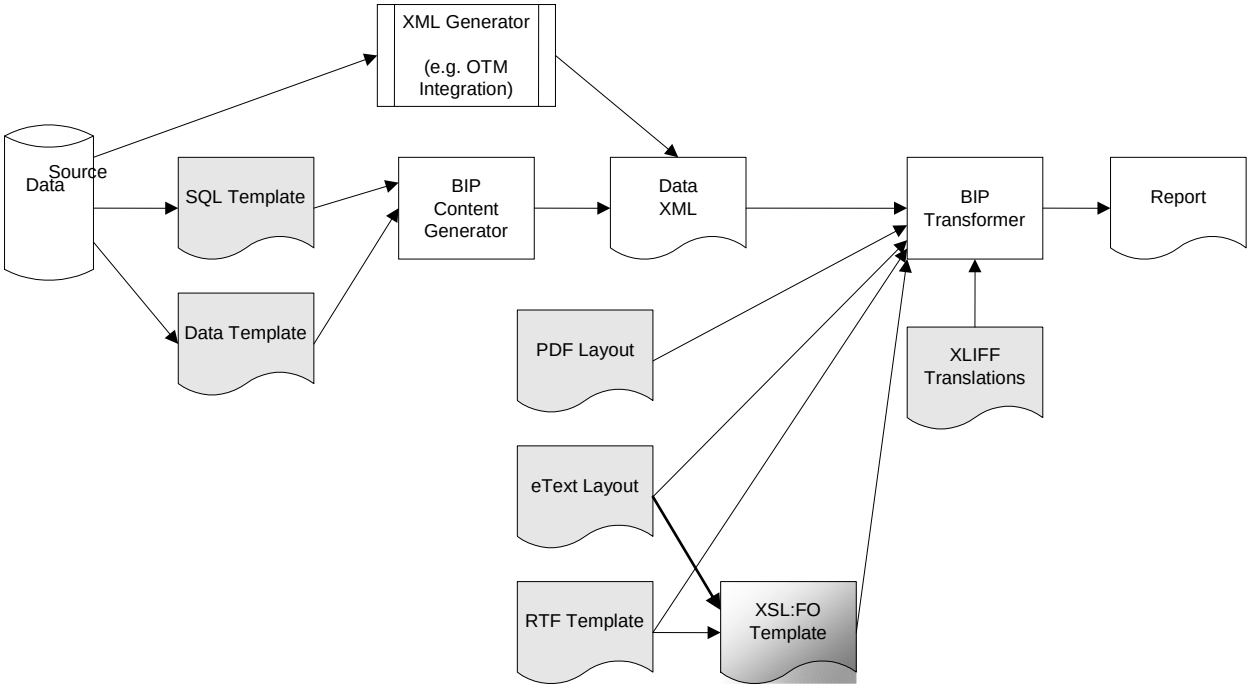


Figure 2: BI Publisher Report Generation

Data Generation

At its core, BI Publisher takes an XML representation of data and transforms it against a template of the report layout. The XML can be generated by:

- A SQL Template. BI Publisher executes the SQL statement against a given data source and generates XML elements for each selected column. Nested elements can be added via cursors. Report parameters are specified with named bind variables. The use of SQL has the advantage of simplicity, but may restrict data needed for layout design.
- A Data Template. A data template is a standardized XML document which defines the information needed to generate data content for the report. It includes a list of report parameters, SQL statements to retrieve content based on those parameters, and a mapping of resulting SQL columns to XML elements. Creating a data template requires understanding of the BI Publisher Data Template schema, but provides greater flexibility by mapping dependent and independent relational data to XML.
- An external XML generator. When using embedded generation, BI Publisher can accept an external XML for data transformation. This approach minimizes data design but may complicate report layout. Specifying individual fields within the report can require complex XPath expressions.

Other data models can be used with BI Publisher, but these are not supported by embedded generation.

Transformation

BI Publisher includes a transformation engine that applies XML data against a report layout. The layout defines the visual structure of the report, including markup tags to populate the report XML from data generation. This markup defines:

- placeholders for XML data. For simple XML schemas, the placeholder can simply be the element name. Complex schemas may require XPath expressions to navigate to the correct data.
- grouping definitions. Tabular data can be inserted into reports using special markup tags to define the repeating XML element. Columns are then defined with placeholders relative to the parent element.
- layout-specific commands.

The output of the transformation engine is a report document. Depending on the layout type, the report user can select the report format.

Table 1 lists each layout supported by BI Publisher, mechanisms for adding markup, and a list of report formats available[2].

Layout Type	Recommended Design Tool	Markup Mechanism	Supported Report Formats
Rich Text Format (RTF)	Microsoft Word with the BI Publisher Template Builder plug-in	XML-style markup tags Help text in MS-Word fields	HTML, RTF, PDF, Excel, CSV[3]
Portable Document Format (PDF)	Adobe Acrobat	Acrobat form fields	PDF
Extensible Stylesheet Language Formatting Objects (XSL:FO)	Any XSL or text editor	XSL transformation commands (e.g. `<xsl:value-of …>`)	HTML, RTF, PDF, Excel, CSV Text, if the XSL stylesheet outputs fixed-column text.
Electronic Text (EFT or EDI)	Microsoft Word with the BI Publisher Template Builder plug-in	XML-style markup tags	Text

Table 1: Report Layouts

Few report designers directly use XSL:FO layouts. For RTF templates, though, transformation performance can be enhanced by converting each RTF layout to XSL:FO. See the **XSL vs. XSL:FO Transformation** section in the **Advanced Layout** chapter for more information.

[2] Oracle Transportation Management v6.2 is certified with BI Publisher v10.1.3.4. BI Publisher v11.1 supports a proprietary layout format for use with its custom layout editor. Report designers can use this layout to develop external reports. Embedded reports will support additional v11.1 layouts in future Oracle Transportation Management releases.

[3] Note that CSV support is available only when BI Publisher is accessed via an external server or farm.

Localization

BI Published supports localization of RTF labels via a translation template. This template is an XML Localization Interchange File Format (XLIFF) file containing translations required by the layout. These are used during transformation for proper translations to be used. An XLIFF is a standard format, which has its own standards and specifications. They can be created manually or automatically generated from the RTF layout. XLIFF files are not required if the report layout is already translated for the locale where it is being used.

Additional Information

For more detailed information regarding data generation and transformation, including details on RTF, PDF and eText markup, consult the Oracle Business Intelligence Publisher Report Designer's Guide.

2. Report Design

This section provides guidelines for designing a report integrated with Oracle Transportation Management. It is not meant to be an exhaustive description of BI Publisher or any other reporting tool, but to present a streamlined approach to quickly build reports. It requires installation and a basic understanding of both products.

Oracle Transportation Management includes a set of analytic and transactional[4] reports available to all report users. As these reports have particularly complex data templates[5], designers are discouraged from using them as templates for custom reports. The guidelines and examples provided in this document should be used as a starting point for new reports.

To create an embedded BI Publisher report, the report designer:

1. Creates a template for data generation. See the **Query Template Creation** section.
2. Creates a template for the report layout. See the **Format Template Creation** section.
3. Creates a report associated with both templates and defines user access to the report. See the **Report Classification** section.
4. Registers custom parameters passed from Oracle Transportation Management to the report. See the **Parameterization** section.

To create an external report, the report designer:

1. Provides a URL to request the report via HTTP.
2. Creates a report associated with the external URL and defines user access to the report. See the **Report Classification** section.
3. Registers custom parameters passed from Oracle Transportation Management to the report. See the **Parameterization** section.

Query Template Creation

A Query Template defines how BI Publisher generates data XML for the report. Oracle Transportation Management supports three types of templates for embedded reports: SQL Templates, Data Templates, and Integration Templates.

SQL Template

A SQL Template is a single SQL statement returning data needed by the report. BI Publisher maps the result set to XML as follows:

- The overall result set is contained in a `<ROWSET>…</ROWSET>` element.
- Each row in the result set is contained in a `<ROW>…</ROW>` element.
- The value of each column in the result set is contained in an element named for the column alias.
- For each nested cursor, BI Publisher outputs the nested result set in an element named for the cursor alias.

[4] A transactional report is one that is focused on a single business object. A shipment document, such as a Bill of Lading, is an example of a transactional report.

[5] This is primarily due to the migration of these reports from Oracle Reports to BI Publisher in Oracle Transportation Management v6.0.

- Each row in the nested result set is contained in a `<cursor alias_ROW>…</cursor alias_ROW>` element.
- The value of each column in the nested result set is contained in an element named for the column alias.

Consider the following SQL example:

```
select s.shipment_gid as gid, s.shipment_xid as xid,
s.shipment_name as name, s.domain_name as domain,
   cursor(
      select distinct
         sip.involved_party_qual_gid as ip_qual,
         sip.involved_party_contact_gid as ip_contact,
      from shipment_involved_party sip
   ) as involved_party
   from shipment s
   where s.shipment_gid = :P_SHIPMENT_ID
```

This query selects the shipment header and a list of involved parties for a given shipment. The resulting XML could be:

```
<?xml version="1.0" encoding="UTF-8"?>
<ROWSET>
   <ROW>
      <GID>MIKEE.000466</GID>
      <XID>000466</XID>
      <NAME>TEST112</NAME>
      <DOMAIN>MIKEE</DOMAIN>
      <INVOLVED_PARTY>
         <INVOLVED_PARTY_ROW>
            <IP_QUAL>CONSIGNEE</IP_QUAL>
            <IP_CONTACT>ESL</IP_CONTACT>
         </INVOLVED_PARTY_ROW>
         <INVOLVED_PARTY_ROW>
            <IP_QUAL>LOGISTICS</IP_QUAL>
            <IP_CONTACT>MIKEE.MAB</IP_CONTACT>
         </INVOLVED_PARTY_ROW>
      </INVOLVED_PARTY>
   </ROW>
</ROWSET>
```

Report designers must adhere to the following rules when developing SQL templates:

1. Every result column and cursor must be uniquely aliased.
2. The alias must adhere to XML element name semantics.
3. All bind variables must be named and map to a standard or custom report parameter passed by Oracle Transportation Management.

Data Template

A Data Template is an XML file specifying parameters and queries needed by the report, along with a mapping of result set data to its XML representation. It provides greater flexibility for:

- parameter specification. Parameters can be typed and explicitly listed.

- data querying. Multiple SQL statements can be specified, allowing independent queries to populate a report.
- data mapping. The results of each SQL statement can be combined, nested, or flattened to simplify the resulting XML, and resulting field specification in the report layout.

There are three primary components of a data template:

1. A list of parameters under a `<parameters>` element. Each parameter is named and typed.
2. A list of SQL data queries under a `<dataQuery>` element. Each SQL query is entered in a `<sqlStatement>` element with a distinct name and a valid SQL query embedded in a `CDATA` section. Note that parameters are available as named bind variables for each query.
3. A tree of XML groups under a `<dataStructure>` element. Each `<group>` element defines a list of XML elements for a particular result set. Each element maps a result set column name or alias to an XML element name.

Consider the following data template example:

```
<dataTemplate name="SIMPLE_SHIPMENT_TEST" description="Simple Shipment Test">
   <parameters>
      <parameter name="P_SHIPMENT_ID" dataType="character"/>
   </parameters>
   <dataQuery>
      <sqlStatement name="SHIPMENT">
         <![CDATA[
            select s.shipment_gid as gid, s.shipment_xid as xid,
                   s.shipment_name as name, s.domain_name as domain
               from shipment s
               where s.shipment_gid = :P_SHIPMENT_ID
            ]]>
         </sqlStatement>
         <sqlStatement name="INVOLVED_PARTY">
            <![CDATA[
               select distinct
                      sip.involved_party_qual_gid as ip_qual,
                      sip.involved_party_contact_gid as ip_contact
               from shipment_involved_party sip
               where sip.shipment_gid = :P_SHIPMENT_ID
            ]]>
         </sqlStatement>
   </dataQuery>
   <dataStructure>
      <group name="SHIPMENT" source="SHIPMENT">
         <element name="GID" value="GID"/>
         <element name="XID" value="XID"/>
         <element name="NAME" value="NAME"/>
         <element name="DOMAIN" value="DOMAIN"/>
         <group name="INVOLVED_PARTY" source="INVOLVED_PARTY">
            <element name="IP_QUAL" value="IP_QUAL"/>
            <element name="IP_CONTACT" value="IP_CONTACT"/>
         </group>
      </group>
   </dataStructure>
</dataTemplate>
```

This data model selects the shipment header and a list of involved parties for a given shipment. The resulting XML could be:

```
<?xml version="1.0" encoding="UTF-8"?>
```



```
<SIMPLE_SHIPMENT_TEST>
   <P_SHIPMENT_ID>MIKEE.000466</P_SHIPMENT_ID>
   <LIST_SHIPMENT>
      <SHIPMENT>
         <GID>MIKEE.000466</GID>
         <XID>000466</XID>
         <NAME>TEST112</NAME>
         <DOMAIN>MIKEE</DOMAIN>
         <LIST_INVOLVED_PARTY>
            <INVOLVED_PARTY>
               <IP_QUAL>CONSIGNEE</IP_QUAL>
               <IP_CONTACT>ESL</IP_CONTACT>
            </INVOLVED_PARTY>
            <INVOLVED_PARTY>
               <IP_QUAL>LOGISTICS</IP_QUAL>
               <IP_CONTACT>MIKEE.MAB</IP_CONTACT>
            </INVOLVED_PARTY>
         </LIST_INVOLVED_PARTY>
      </SHIPMENT>
   </LIST_SHIPMENT>
</SIMPLE_SHIPMENT_TEST>
```

Note that, unlike SQL Templates, all parameters are included in the data XML. This can be useful in layout design.

Detailed information regarding Data Template design can be found in the BI Publisher documentation: Building a Data Template.

Integration Template

An Integration Template is a special query template that relies on the Oracle Transportation Management integration layer to generate data XML for BI Publisher layouts. It is reserved for transactional reports. The XML schema for each transportation business object can be found in the Oracle Transportation Management Integration Guide.

Though an integration template is the simplest query template to specify, it generates complex data XML. This can complicate layout design as full XPath expressions may be needed for each data field. See the **Navigating Oracle Transportation Management Integration XML** section for more information.

Uploading a Query Template for Embedded Reporting

After selecting and generating query template content, the report designer needs to upload the query template for use in Oracle Transportation Management. They should:

1. Log into Oracle Transportation Management. They must have rights to create report data.
2. Navigate to **Business Process Automation > Power Data > Document Generation > Query Template.**
3. Create a new query template.
4. If defining a SQL or data template, upload the respective `.sql` or `.xml` file to the system. If the report is transactional, select the associated Data Query Type. This will add a check to determine that the data template, format template, and report types all match.
5. If defining an integration template, select the associated Data Query Type. Optionally, select an Out XML Profile to minimize the generated XML as needed.

6. Generate sample XML for layout design. This action will prompt you for any required template parameters:
   - for a SQL template, this includes all named bind variables
   - for a Data template, this includes all entries in the `<parameter>` section
   - for an Integration template, this is the GID of the business object

   Note: This XML can be used to define the XML data model for the BI Publisher Microsoft Word add-in . To best leverage this tool, the data source should populate all possible fields in the XML. Otherwise, the designer may have to directly insert XPath markup into his layout document.

7. Save the generate XML to disk.

Format Template Creation

Once a query template is defined, the report designer creates a Format Template. The Format Template is a definition of both the report layout and any XLIFF translations to apply to it. Table 1 lists layouts supported by Oracle Transportation Management and the recommended tools for generating them.

Though designers are free to use all features of the BI Publisher Microsoft Word add-in or other tools to generate layout, the following steps are recommended for RTF report designers using Oracle Transportation Management:

- Use Microsoft Word with the BI Publisher Microsoft Word add-in to design RTF templates.
- Import the sample XML generated in step 6 above via `Data > Load XML Data`. This avoids the need for per-report schemas, defining data templates within BI Publisher Enterprise, and a .NET/J2EE connection between Microsoft Word and BI Publisher.
- Use `Input > Field` to add simple data fields to the layout. If a field is not uniquely named in the XML schema, edit the field properties and add distinguishing XPath.
- Use `Input > Table Wizard` to add tabular data fields to the layout.
- Save the layout as an `RTF` and use `Tools > Export > XSL-FO Style Sheet` to export the layout as an XSL:FO file. This is for performance optimization. Though the RTF file can be uploaded as a layout to Oracle Transportation Management, each generation of the report will require transformation of the RTF to an XSL:FO. By uploading the transformed XSL:FO to the system, report generation time can be reduced.

Uploading a Format Template for Embedded Reporting

After generating a layout, the report designer needs to upload it to a format template for use in Oracle Transportation Management. They should:

1. Log into Oracle Transportation Management. They must have rights to create report data.
2. Navigate to **Business Process Automation > Power Data > Document Generation > Format Template.**
3. Create a new format template.
4. Upload the layout to the system. This should be a .pdf, .rtf or .xsl file, depending on the layout type.
5. If the layout represents an electronic text specification, select the **eText Template** check box.
6. If the layout represents a BI Publisher stylesheet, select the **XSL-FO Template** check box. If this check box is cleared, the system transforms XML data using a simple Xalan XSLT transformation and circumvents BI Publisher entirely.
7. If the report is transactional, select the associated Data Query Type. This enforces type checking against query templates and report definitions.

8. Add **Translation Template** records for each XLIFF file supported by the report.

Report Classification

With query and format templates defined, the designer then classifies the report. They should:

1. Log into Oracle Transportation Management. They must have rights to create report data.
2. Navigate to **Business Process Automation > Power Data > Document Generation > Reports**.
3. Create a new report. By default, the report will be created in your domain.
4. Select the **Select Via UI** check box. This allows the report to be explicitly run by Oracle Transportation Management users.
5. Select the **Use Report Parameters as Bind Values** check box. See the **Dynamic SQL Parameterization** section for more information on this option.
6. Follow the flowchart in Figure 3 to determine how the report should be accessed.

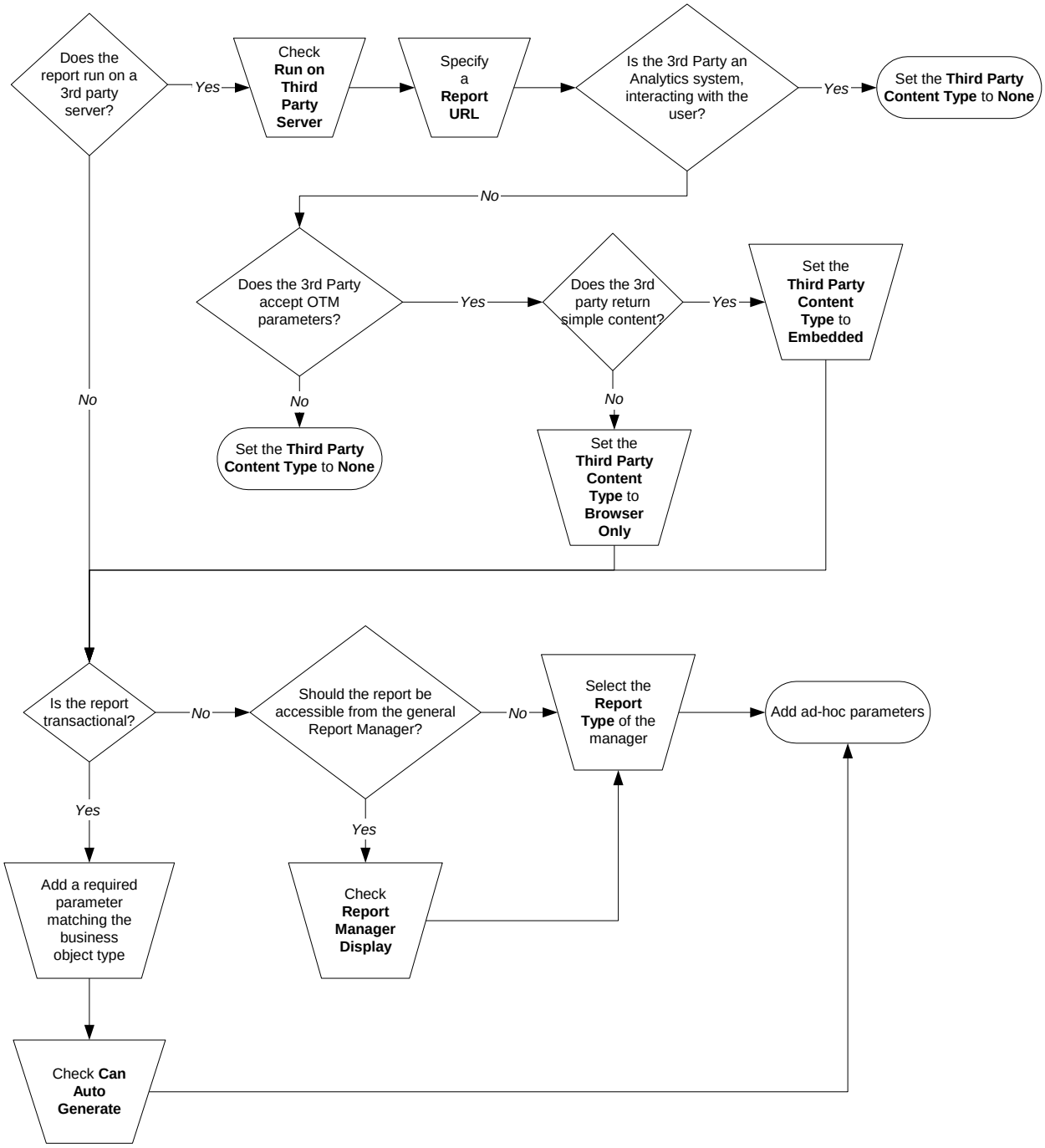


Figure 3: Report Classification

Parameterization

For both embedded and external reports, Oracle Transportation Management passes standard and ad hoc parameters to the report generator[6]. Table 2 summarizes the standard parameters passed to every report.

Name	Description	Comments
P_REPORT_GID	The requested report	External systems are responsible for mapping the Oracle Transportation Management report GID to a valid report. Alternatively, the report URL can embed the mapping within a request parameter.
P_DBCONN_TYPE	The data schema	OLTP for the Oracle Transportation Management transactional database; ODS for the offline analytical database.
P_GL_USER	The user requesting the report	This may be used by external systems to enforce VPD data security on report queries.
P_ROLE_ID	The role of the requesting user	This may be used by external systems to enforce VPD data security on report queries.
P_DOMAIN	The domain of the requesting user	
P_LANGUAGE	The ISO language code requested for the report	For embedded reports, XLIFF translations are automatically applied.
P_COUNTRY	The ISO country code requested for the report	For embedded reports, XLIFF translations are automatically applied.
P_DISPLAY_NAME	A user-readable name for the report.	
P_DATE_FORMAT	The Oracle date format for both input parameters and output fields.	For input parameters, use the `TO_DATE(:P_MY_DATE, :P_DATE_FORMAT)` function. For output parameters, use the `TO_CHAR(field, :P_DATE_FORMAT)` function. This should be reserved for fields that are date-only.

[6] If external report generators do not support these parameters, their content type should be set to **None**.

Name	Description	Comments
P_DATE_TIME_FORMAT	The Oracle timestamp format for both input parameters and output fields.	For input parameters, use the `TO_DATE(:P_MY_TIMESTAMP, :P_DATE_TIME_FORMAT)` function. For output fields, use the `TO_CHAR(field, :P_DATE_TIME_FORMAT)` function. This should be reserved for fields that have both date and time.

Table 2: Standard Parameters

The report designer may add additional, ad-hoc parameters to a report using the Report Parameters grid of the Report Manager. Before submitting a report request, you are presented with a list of these additional parameters and must enter information for any marked as **Mandatory**.[7]

For BI Publisher, all parameters are available for data generation and layout as follows:

- **SQL Template**: Each parameter is mapped into a bind variable matching the Report Parameter Name with a string value. If any bind variable in the SQL template is not specified in the report parameters, BI Publisher fails to generate data.
- **Data Template**: Each parameter is mapped to a `<parameter>` entry in the data template. The parameter `name` attribute must match the Report Parameter Name or the parameter is ignored. All values are strings. If any template parameter is not specified in the report parameters, BI Publisher fails to generate data.
- **Layout**: If the XML data came from a Data Template, all parameters are passed as top-level elements[8]. These can be used within the layout. Otherwise, parameters are not available.

A transactional report is defined as a report with a mandatory parameter matching a particular business object type.

For external reports, parameters are added as HTTP request parameters to the URL.

[7] Note that transactional reports have at least one mandatory parameter matching the business object type. If requested from a transactional manager, the system automatically populates this parameter. E.g., a shipment document with one mandatory **`P_SHIPMENT_ID`** parameter with a **`Query Name`** of **`BUY_SHIPMENT`** can be selected from the shipment manager. The **`P_SHIPMENT_ID`** parameter is automatically populated with the relevant shipment GID.

[8] This is default behavior. The BI Publisher **`include_parameters`** property in the data template can be set to false to suppress it.

3. Tutorials

Simple Shipment Summary

If you want to generate a simple summary of shipments whose start time is between a date range that you specify. The summary should include shipment header information like XID, Name, Start Time, End Time, and Domain; along with a list of involved parties.

The basic steps to create an embedded analytical report are:

1. Generate a data model. For this report, you will use a SQL Template.
2. Create a Query Template record within the system and upload the data model to it.
3. Export sample XML data for layout design.
4. Design a layout using the XML fields. For this report, you will define an RTF layout using the BI Publisher Template Builder.
5. Create a Format Template record within the system and upload the layout to it.
6. Create a Report record within the system, mapping the query template from #2 and the format template from #5. You will also specify any custom parameters needed by the report and select a Report Manager UI to hold the report links.
7. Test the report.

Step 1: Generate a Data Model

The following is a SQL Template for the report. It is located in:

```
<OTM_INSTALL_DIR>/samples/reports/SimpleShipmentSummary/SimpleShipmentSummary.s
ql

select
   :P_EARLIEST_DATE as earliest,
   :P_LATEST_DATE as latest,
   cursor(select
      s.shipment_xid as xid,
      s.shipment_name as name,
      to_char(s.start_time, :P_DATE_TIME_FORMAT) as start_time,
      to_char(s.end_time, :P_DATE_TIME_FORMAT) as end_time,
      s.domain_name as domain,
      cursor(select distinct
         sip.involved_party_qual_gid as ip_qual,
         sip.involved_party_contact_gid as ip_contact
      from shipment_involved_party sip
      where sip.shipment_gid=s.shipment_gid
      order by sip.involved_party_qual_gid) as involved_party
   from shipment s
   where s.start_time >= to_date(:P_EARLIEST_DATE, :P_DATE_FORMAT)
      and s.start_time <= to_date(:P_LATEST_DATE, :P_DATE_FORMAT)
   order by s.start_time, s.shipment_gid
   ) as shipment
from dual
```

Note that:

- Every select column and cursor has an alias. This alias is used to define XML elements and must adhere to element naming restrictions.

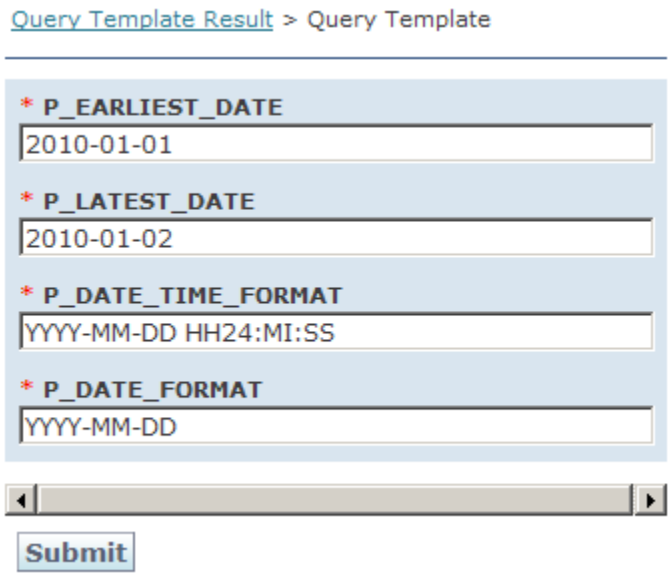
- The bind variables implicitly define four parameters for the report: `P_EARLIEST_DATE`, `P_LATEST_DATE`, `P_DATE_TIME_FORMAT`, and `P_DATE_FORMAT`. The last two parameters are provided by the system to every report. The first two are custom parameters.
- To use SQL Template parameters in the layout, you must select the parameters from `dual`. The outermost `select` defines a single `<ROW>` for parameters.
- The first nested cursor selects shipment header information. Start and end time are formatted with `P_DATE_TIME_FORMAT` to reflect user preferences. Each matching shipment is mapped to a `<SHIPMENT_ROW>` based on the cursor alias.
- The system passes all parameters as text. In particular, dates and timestamps are formatted using user preference date or date/time formats. To compare dates, you need to convert `P_EARLIEST_DATE` and `P_LATEST_DATE` to dates via the `to_date()` function.
- The second nested cursor selects involved parties for each shipment. Each party is mapped to an `INVOLVED_PARTY_ROW` based on the cursor alias.

Step 2: Create a Query Template

1. Log into Oracle Transportation Management and navigate to **Business Process Automation > Power Data > Document Generation > Query Template**. Create a new query template and upload the `sql` file.
2. Select **Finished** to save the query template.

Step 3: Export Sample XML for Layout Design

1. Reopen the query template and select **Generate Sample XML**. This will prompt you for the four bind variables:



2. Fill in as follows:
   - `P_EARLIEST_DATE`. The earliest start time to retrieve shipments. This should be in `P_DATE_FORMAT` (`YYYY-MM-DD`).
   - `P_LATEST_DATE`. The latest start time to retrieve shipments. This should be in `P_DATE_FORMAT` (`YYYY-MM-DD`).
   - `P_DATE_TIME_FORMAT = YYYY-MM-DD HH24:MI:SS`
   - `P_DATE_FORMAT = YYYY-MM-DD`

3. The date range should be picked to produce 10-50 shipments. Once you select **Submit**, your browser will prompt you to open or save the document.
4. Save the document to a drive accessible by the BI Publisher Template Builder.
5. Close the Generate Sample XML window.

Step 4: Design a Layout

To modify the sample layout file for the report:

1. Open up Microsoft Word with the BI Publisher Template Builder.
2. Open up `<OTM_INSTALL_DIR>/samples/reports/SimpleShipmentSummary/SimpleShipmentSummary.rtf` holding the layout. Your layout will resemble:

3. Select **Data > Load XML Data** from the BI Publisher Template Builder menu and select the XML file saved in Step 3 above. This associates the sample XML which is implicitly defining an XML schema for the layout.
4. Use **Insert > Field and Insert > Table Wizard** to add ad-hoc and tabular data to the layout.
5. Use **Preview** to test the modified layout with the sample XML.
6. Save the modified layout as a Rich Text Format document to a drive accessible by your local browser. Do not overwrite the sample layout.
7. Optionally, use **Tools > Export > XSL:FO Style Sheet** to export the layout as an XSL:FO document. Save the displayed document with an `xsl` extension to a drive accessible by your local browser. Using an XSL layout will improve report performance.

Step 5: Create a Format Template

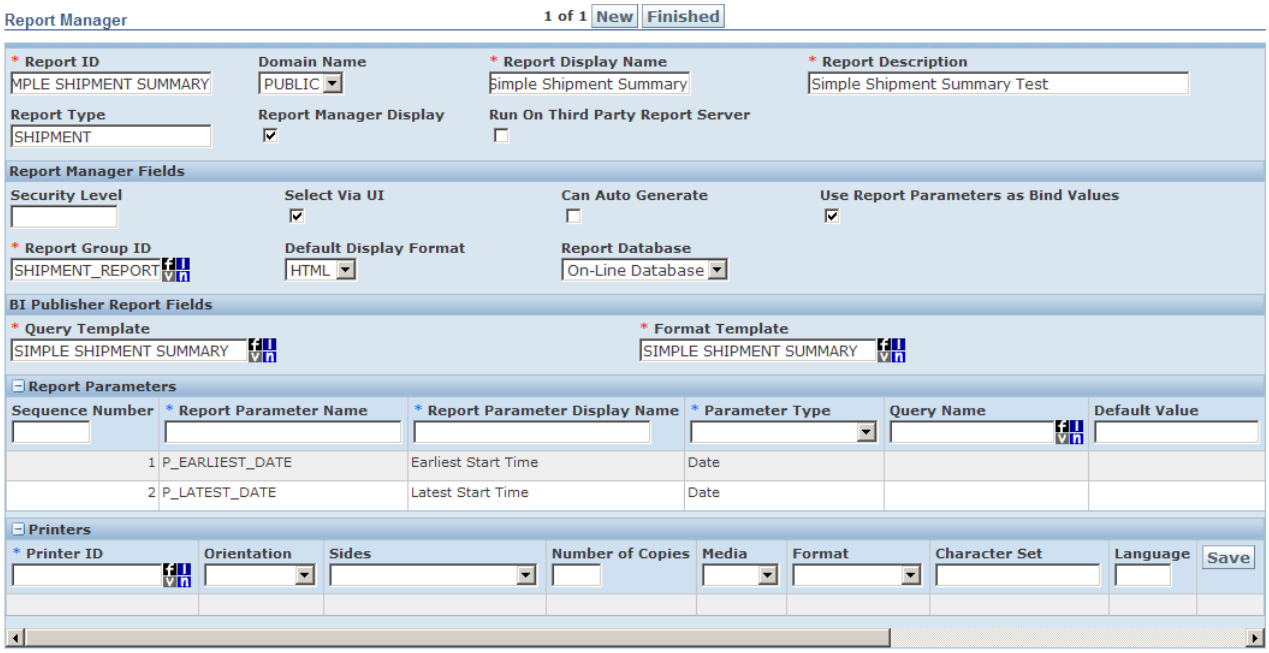
1. Return to Oracle Transportation Management and navigate to **Business Process Automation > Power Data > Document Generation > Format Template**. Create a new format template and either:
   - Upload the RTF file created in step 4; or
   - Upload the XSL file created in step 4 and select the **XSL-FO Template** check box.
2. Select **Finished** to save the format template.

Step 6: Create a Report

1. Navigate to **Business Process Automation > Power Data > Document Generation > Reports**. Create a new report as follows:
2. Set **Report ID** to *SIMPLE SHIPMENT SUMMARY*.
3. Set **Report Display Name** to *Simple Shipment Summary*.
4. Set **Report Description** to *Simple Shipment Summary Test*.
5. Set **Report Type** to *SHIPMENT*. The report should be available on the Reports menu under Shipment Management.

6. Select the **Report Manager Display** check box. The report should be available under **Business Process Automation > Reporting > Report Manager**.
7. Select the **Use Report Parameters as Bind Values** check box.
8. Select the **Select Via UI** check box.
9. Set **Report Group ID** to *SHIPMENT_REPORTS*. In the overall report manager, the report should be grouped with other shipment reports.
10. Set the **Query Template** to *SIMPLE SHIPMENT SUMMARY*.
11. Set the **Format Template** to *SIMPLE SHIPMENT SUMMARY*.
12. Add two mandatory date parameters: *P_EARLIEST_DATE* and *P_LATEST_DATE*.

The resulting report in the Report Manager is shown below:



13. Select **Finished** to save the report.

Step 7: Test the Report

1. Navigate to **Shipment Management > Reports** and find the `Simple Shipment Summary` grouped with the `Shipment Reports`. Select **Run Online** to be prompted for custom parameters and delivery information.
2. Enter the earliest and latest start times and **Submit** to test the report. A PDF output of the report will resemble:

Simple Shipment Summary

Shipments from 2010-01-01 to 2010-01-02

ID	Name	Start	End	Domain	Involved Parties	
01552		2010-01-01 00:39:00	2010-01-02 00:00:35	CONNIE		
01553		2010-01-01 00:39:00	2010-01-05 07:53:00	CONNIE		
03871		2010-01-01 12:00:00	2010-01-01 12:20:46	LISA		
03871		2010-01-01 12:00:00	2010-01-01 12:20:46	LISANAT		
03871		2010-01-01 12:00:00	2010-01-01 12:20:46	NATNAL		
03871		2010-01-01 12:00:00	2010-01-01 12:20:46	SK1		
TEST	TEST	2010-01-01 13:00:00	2010-01-01 13:30:00	PRFDOCK		
60618		2010-01-01 14:00:00	2010-01-02 20:11:00	DGANO		
03519		2010-01-01 14:00:00	2010-01-02 02:00:00	LISA		
03678		2010-01-01 14:00:00	2010-01-02 02:00:00	LISA		
03838		2010-01-01 14:00:00	2010-01-01 23:30:00	LISA		
03519		2010-01-01 14:00:00	2010-01-02 02:00:00	LISANAT		
03678		2010-01-01 14:00:00	2010-01-02 02:00:00	LISANAT		
03838		2010-01-01 14:00:00	2010-01-01 23:30:00	LISANAT		
03519		2010-01-01 14:00:00	2010-01-02 02:00:00	NATNAL		
03678		2010-01-01 14:00:00	2010-01-02 02:00:00	NATNAL		
03838		2010-01-01 14:00:00	2010-01-01 23:30:00	NATNAL		
03519		2010-01-01 14:00:00	2010-01-02 02:00:00	SK1		
03678		2010-01-01 14:00:00	2010-01-02 02:00:00	SK1		
03838		2010-01-01 14:00:00	2010-01-01 23:30:00	SK1		
60618		2010-01-01 14:00:00	2010-01-02 20:11:00	ZOMI		
11750		2010-01-01 14:30:00	2010-01-02 07:00:00	GUEST	BILL-TO	GUEST.ADMIN
01354		2010-01-01 17:57:00	2010-01-02 20:48:00	SHOWTELL/TL	BILL-TO CUSTOMER SHOWTELL LI ABLE PARTY	SHOWTELL.USB ANK SHOWTELL.HQ Q21 SHOWTELL.HQ Q21
12311		2010-01-01 20:30:00	2010-01-02 03:00:00	GUEST	BILL-TO	GUEST.ADMIN
10 DATA1		2010-01-01 21:51:00	2010-02-28 21:51:00	ILA	BILL-FROM BILL-TO LOGISTICS REMIT-TO	LAX LAX LAX LAX
03494		2010-01-01 23:00:00	2010-01-02 01:00:00	LISA		
03494		2010-01-01 23:00:00	2010-01-02 01:00:00	LISANAT		
03494		2010-01-01 23:00:00	2010-01-02 01:00:00	NATNAL		
03494		2010-01-01 23:00:00	2010-01-02 01:00:00	SK1		

Simple Commercial Invoice

Assume you want to generate a Commercial Invoice report for a given shipment. The report should include shipment header information such as: Seller, Consignee, Buyer, Origin, Destination, Weight, Volume, and Cost. It should also list the contents of the items on the shipment.

The basic steps to create an embedded transactional report are:

1. Generate a data model. For this report, you will use a **Data Template**.
2. Create a **Query Template** record within the system and upload the data model to it.
3. Export sample XML data for layout design.
4. Design a layout using the XML fields. For this report, you will define an RTF layout using the BI Publisher Template Builder.
5. Create a **Format Template** record within the system and upload the layout to it.
6. Create a **Report** record within the system, mapping the query template from #2 and the format template from #5, specifying a single mandatory shipment parameter needed by the report.
7. Test the report via the Shipment Manager action menu.

Step 1: Generate a Data Model

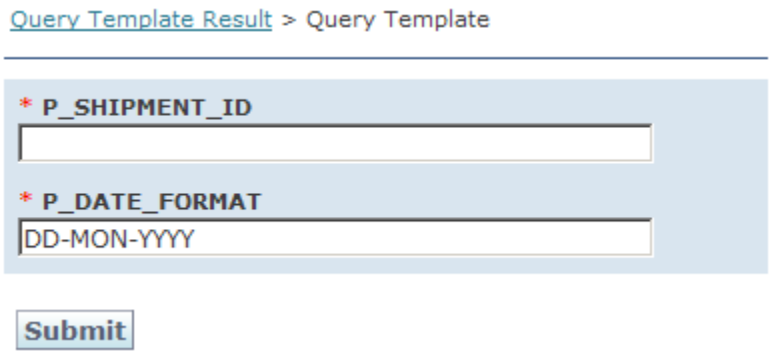
1. The Data Template for the report can be found in `<OTM_INSTALL_DIR>/samples/reports/SimpleCommercialInvoice/SimpleCommercialInvoice.xml`. Note that:
   - The report has a single parameter, `P_SHIPMENT_ID`. This will be defined as a mandatory parameter with a query table of `SHIPMENT` to mark the report as a transactional shipment report.
   - The report will hold two SQL statements. The first, `q_shipment`, retrieves header information based on the `:P_SHIPMENT_ID` bind variable. The results are mapped to a top level `<G_SHIPMENT>` XML element.
   - The second SQL statement, `q_items`, retrieves item information associated with the shipment. It basis its query on the first XML element `<SHIPMENT>` defined in `<G_SHIPMENT>`. BI Publisher automatically maps this element to a `:SHIPMENT` bind variable. While `q_items` could simply have reused the `:P_SHIPMENT_ID` parameter, this shows the implicit relationships between disparate queries. The item results are stored in nested `<G_ITEM>` elements under `<G_SHIPMENT>`.
   - This report makes significant use of PL/SQL utilities provided with the system. See the **Using Utility Packages** section for more information

Step 2: Create a Query Template

1. Log into Oracle Transportation Management and navigate to **Business Process Automation > Power Data > Document Generation > Query Template**. Create a new query template, set the **Data Query Type** to *SHIPMENT* and upload the xml file.
2. Select **Finished** to save the query template.

Step 3: Export Sample XML for Layout Design

1. Reopen the query template and click **Generate Sample XML**. This will prompt you for the report parameters, including a sample shipment GID:



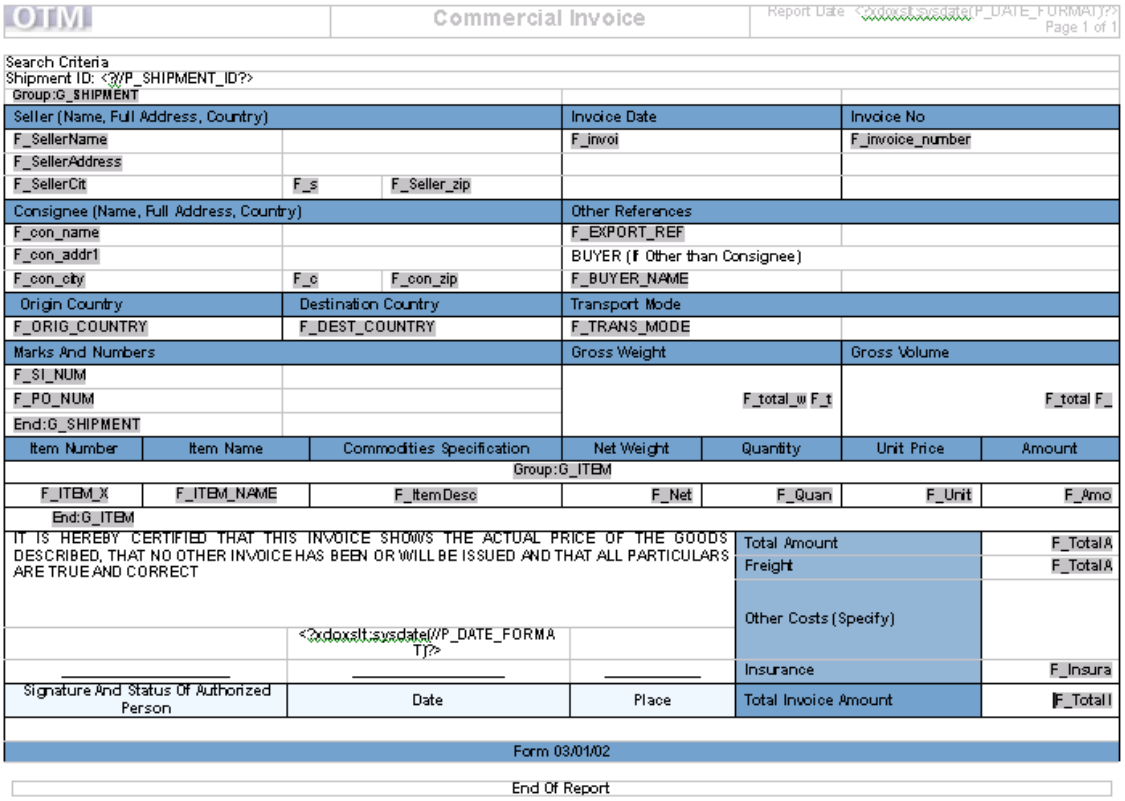
2. Select a shipment that has a seller, consignee, buyer, weight, volume, cost, and distinct items. Once you click **Submit**, your browser will prompt you to open or save the document.
3. Save the document to a drive accessible by the BI Publisher Template Builder.
4. Close the Generate Sample XML page.

Step 4: Design a Layout

To modify the sample layout file for the report:

1. Open up Microsoft Word with the BI Publisher Template Builder

2. Open `<OTM_INSTALL_DIR>/samples/reports/SimpleCommercialInvoice/SimpleCommercialInvoice.rtf` holding the layout. Your layout will resemble:



OTM | Commercial Invoice | Report Date: <?xdoxslt:sysdate(P_DATE_FORMAT)?> Page 1 of 1

Search Criteria
Shipment ID: <?P_SHIPMENT_ID?>
Group:G_SHIPMENT

Seller (Name, Full Address, Country)			Invoice Date	Invoice No
F_SellerName			F_invoi	F_invoice_number
F_SellerAddress				
F_SellerCit	F_s	F_Seller_zip		
Consignee (Name, Full Address, Country)			**Other References**	
F_con_name			F_EXPORT_REF	
F_con_addr1			BUYER (If Other than Consignee)	
F_con_city	F_c	F_con_zip	F_BUYER_NAME	
Origin Country	**Destination Country**		**Transport Mode**	
F_ORIG_COUNTRY	F_DEST_COUNTRY		F_TRANS_MODE	
Marks And Numbers			**Gross Weight**	**Gross Volume**
F_SI_NUM			F_total_w F_t	F_total F_
F_PO_NUM				
End:G_SHIPMENT				

Item Number	Item Name	Commodities Specification	Net Weight	Quantity	Unit Price	Amount
Group:G_ITEM						
F_ITEM_X	F_ITEM_NAME	F_ItemDesc	F_Net	F_Quan	F_Unit	F_Amo
End:G_ITEM						

IT IS HEREBY CERTIFIED THAT THIS INVOICE SHOWS THE ACTUAL PRICE OF THE GOODS DESCRIBED, THAT NO OTHER INVOICE HAS BEEN OR WILL BE ISSUED AND THAT ALL PARTICULARS ARE TRUE AND CORRECT			Total Amount	F_TotalA
			Freight	F_TotalA
			Other Costs (Specify)	
	<?xdoxslt:sysdate(//P_DATE_FORMAT)?>			
_____	_____	_____	Insurance	F_Insura
Signature And Status Of Authorized Person	Date	Place	Total Invoice Amount	F_TotalI

Form 03/01/02

End Of Report

3. Select **Data > Load XML Data** from the BI Publisher Template Builder menu and select the XML file saved in Step 3 above. This associates sample XML, implicitly defining an XML schema for the layout.
4. Use **Insert > Field and Insert > Table Wizard** to add ad hoc and tabular data to the layout.
5. Use **Preview** to test the modified layout with the sample XML.
6. Save the modified layout as a Rich Text Format document to a drive accessible by your local browser. Do not overwrite the sample layout.
7. Optionally, use **Tools > Export > XSL:FO Style Sheet** to export the layout as an XSL:FO document. Save the displayed document with an `xsl` extension to a drive accessible by your local browser. Using an XSL layout will improve report performance.

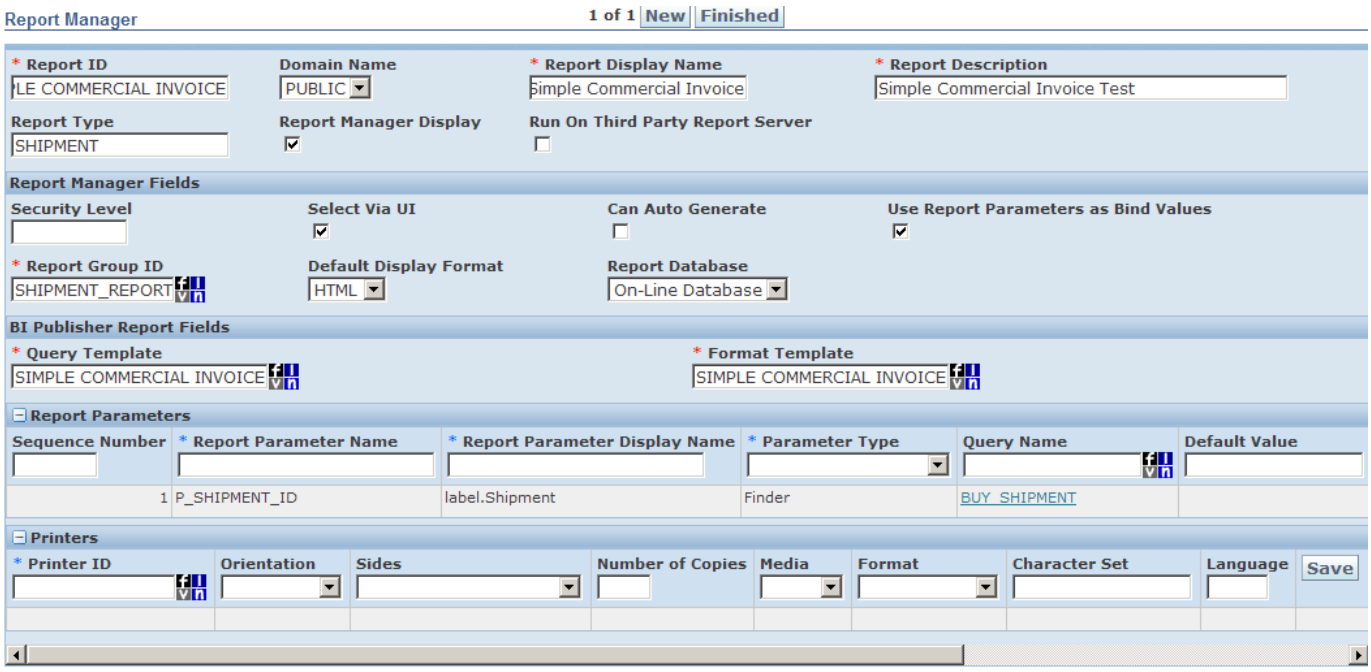
Step 5: Create a Format Template

1. Return to Oracle Transportation Management and navigate to **Business Process Automation > Power Data > Document Generation > Format Template**. Create a new format template and either:
   - upload the RTF file created in step 4; or
   - upload the XSL file created in step 4 and check the **XSL-FO Template** check box.
2. Select **Finished** to save the format template.

Step 6: Create a Report

1. Navigate to `Business` **Process Automation > Power Data > Document Generation > Reports**. Create a new report as follows:
2. Set **Report ID** to *SIMPLE COMMERCIAL INVOICE*.
3. Set **Report Display Name** to *Simple Commercial Invoice*.
4. Set **Report Description** to *Simple Commercial Invoice Test*.
5. Set **Report Type** to *SHIPMENT*. The report should be available on the Reports menu under Shipment Management.
6. Select the **Report Manager Display** check box. The report should be available under **Business Process Automation > Reporting > Report Manager**.
7. Select the **Use Report Parameters as Bind Values** check box.
8. Select the **Select Via UI** check box.
9. Set **Report Group ID** to *SHIPMENT_REPORTS*. In the overall report manager, the report should be grouped with other shipment reports.
10. Set the **Query Template** to *SIMPLE COMMERCIAL INVOICE*.
11. Set the **Format Template** to *SIMPLE COMMERCIAL INVOICE*.
12. Add a mandatory `P_SHIPMENT_ID` parameter. This parameter should have a **Parameter Type** of *Finder* and a **Query Name** of *BUY_SHIPMENT*. When selected from the Shipment Manager actions menu, the system automatically populates the parameter with the selected shipment ID. When selected from the Reports Manager, you will be prompted with a Shipment pick list. Note that the second parameter `P_DATE_FORMAT` is a standard parameter passed to all reports.

The resulting report is:



13. Select **Finished** to save the report.

Step 7: Test the Report

1. Navigate to **Shipment Management > Shipment Management > Buy Shipments**. Query for shipment results and select one of the matching shipments. Select **Actions > Business Process Automation > Reports > Custom Reports**. You are prompted to select a report.
2. Select the *Simple Commercial Invoice*.
3. **Submit** to test the report. A PDF output of the report will resemble:

OTM

Commercial Invoice

Report Date 10/20/2010
Page 1 of 1

Search Criteria
Shipment ID: DGANO.47227

Seller (Name, Full Address, Country)			Invoice Date	Invoice No
HOUSTON NAME POB 1897 WIESS COLLEGE HOUSTON TX			10/20/2010	47227
Consignee (Name, Full Address, Country)			**Other References**	
PEACHTREE ATLANTA GA 30301			**BUYER (If Other than Consignee)**	
Origin Country	**Destination Country**		**Transport Mode**	
UNITED STATES	UNITED STATES		TL	
Marks And Numbers			**Gross Weight**	**Gross Volume**
			40,000.00 LB	2,500.00 CUFT

Item Number	Item Name	Commodities Specification	Net Weight	Quantity	Unit Price	Amount
FROZEN			40,000.00	1.00		123.45

IT IS HEREBY CERTIFIED THAT THIS INVOICE SHOWS THE ACTUAL PRICE OF THE GOODS DESCRIBED, THAT NO OTHER INVOICE HAS BEEN OR WILL BE ISSUED AND THAT ALL PARTICULARS ARE TRUE AND CORRECT

10/20/2010

Total Amount	123.45
Freight	24.69
Other Costs (Specify)	
Insurance	20.58
Total Invoice Amount	168.72

Signature And Status Of Authorized Person	Date	Place

Form 03/01/02

End Of Report

Note: Reports entered in the `PUBLIC` domain can be selected directly off of the actions menu. E.g., if the Simple Commercial Invoice report is PUBLIC, you can navigate to **Actions > Business Process Automation > Reports > Simple Commercial Invoice**.

4. Advanced Content

Date and Timestamp Handling

Report designers should take special care when writing queries or layout involving DATE columns. This includes:

- Converting date parameters for where clause comparison. All date parameters are sent as strings, formatted according to your date preferences. To compare one to a data field, the designer should use Oracle's `TO_DATE` function, applying the standard `P_DATE_FORMAT` parameter:

```
where accessorial_cost.effective_date > TO_DATE(:P_EARLIEST_DATE,
:P_DATE_FORMAT)
    and accessorial_cost.effective_date < TO_DATE(:P_LATEST_DATE,
:P_DATE_FORMAT)
```

- Converting timestamp parameters for where clause comparison. Like dates, timestamp parameters are sent as strings, formatted according to your date/time preferences[9]. To compare one to a timestamp field, the designer should use Oracle's `TO_DATE` function, applying the standard `P_DATE_TIME_FORMAT` parameter:

```
where shipment.start_time > TO_DATE(:P_EARLIEST_START, :P_DATE_TIME_FORMAT)
    and shipment.start_time < TO_DATE(:P_EARLIEST_START,
:P_DATE_TIME_FORMAT)
```

- Applying your date and time preferences to report layout. Depending on the use case, designers may want to apply user preferences to dates displayed on the final report. If so, the query template should convert selected date and timestamp values using Oracle's `TO_CHAR` function:

```
select TO_CHAR(accessorial_cost.effective_date, :P_DATE_FORMAT) …
select TO_CHAR(shipment.start_time, :P_DATE_TIME_FORMAT) …
```

- Accounting for UTC storage in report layout. Nearly all timestamp fields in Oracle Transportation Management are converted to UTC before persisting to the database. A report designer who simply queries `shipment.start_time`, for example, receives the time in UTC. To convert the stored time to the application or report server's time zone, use the `vpd.gmt_offset` function:

```
select TO_CHAR(shipment.start_time-(vpd.gmt_offset/24), :P_DATE_TIME_FORMAT)
…
```

- Accounting for UTC storage in query template criteria. To compare timestamp fields to some offset of current time, designers can either convert the field or use `vpd.gmt_sysdate`. This function returns the current time in UTC. E.g. to query shipments starting in the next three days:

```
select shipment.gid
   where shipment.start_time > vpd.gmt_sysdate
   and shipment.start_time < vpd.gmt_sysdate+3
```

[9] Note that the Reporting has never applied user time preference. Time preference is assumed to be HH24:MI:SS.

Data Security

Oracle Transportation Management implements data security via Oracle's Virtual Private Database (VPD). Given a user and their role, the system limits row access to tables. The default VPD policies implement a domain model, where a user's domain determines their read/write privileges. Specific implementations, however, can enhance this model to restrict access based on other columns and grant access across domains.

When running embedded reports, VPD security is automatically enforced by the system. The database connection passed to the BI Publisher APIs maintains the user and role in context, applying VPD policies on any queries.

External report generators, however, have two options regarding data security:

- Suppress it. The standard `glogdba` database user requires a VPD context. If a report logs in as `glogowner`, however, VPD is suppressed. The report queries have access to all data. This may be appropriate for transactional reports where the data is restricted to a particular business object. Alternatively, each report can implement its own data security model independent of Oracle Transportation Management.
- Set the user context before issuing any queries. The `GLOGOWNER.VPD` package provides the following procedures and functions to set the context for VPD:

```
procedure set_user (user VARCHAR2);
procedure set_user_r (user VARCHAR2, user_role varchar2);

function set_user_fct (user VARCHAR2);
function set_user_r_fct (user VARCHAR2, user_role varchar2);
```

By passing the standard P_GL_USER (and optionally P_ROLE_ID) parameter to one of these functions, VPD returns the proper rows.

When designing, testing, or generating reports directly within the BI Publisher web application, a report designer must choose one of these options as well. The report data source can login as `glogowner` and suppress VPD, though this may return too many rows for analytical reports. If logged in as `glogdba`, setting the context for VPD can be executed in the first query in the data template[10]. For example,

```
<dataTemplate name="vpd_test" dataSourceRef="glogdba_data_source">
    …
    <parameters>
        <parameter name="P_GL_USER" dataType="character"/>
        …
    </parameters>
    <dataQuery>
        <sqlStatement name="VPD">
            <![CDATA[
                  SELECT vpd.set_user_fct(:P_GL_USER) as gl_user from dual
              ]]>
        </sqlStatement>
        <sqlStatement name="Q_1">
            …
        </sqlStatement>
    </dataQuery>
    <dataStructure>
```

[10] SQL Templates do not support VPD except in embedded reports.


```
        …
    </dataStructure>
</dataTemplate>
```

This report prompts for the P_GL_USER and supplies it to the VPD package, prior to making any queries. Though this is ideal when BI Publisher is acting as an external report generator to Oracle Transportation Management (since P_GL_USER is passed as a standard parameter), it doesn't truly enforce security during design or when generating reports through the BI Publisher web interface. You can take this one step farther by leveraging BI Publisher security. Assume BI Publisher users have the same name (case insensitive) as Oracle Transportation Management user GID's. Then the following data template applies full VPD security whether BI Publisher is used for design, ad-hoc generation, or an external generator:

```
<dataTemplate name="vpd_test" dataSourceRef="glogdba_data_source">
    …
    <parameters>
        <parameter name="xdo_user_name" dataType="character"/>
        <parameter name="P_GL_USER" dataType="character"/>
        …
    </parameters>
    <dataQuery>
        <sqlStatement name="VPD">
            <![CDATA[
                SELECT vpd.set user fct(upper(nvl(:P GL USER,:xdo user_name)))
                as gl_user from dual
            ]]>
        </sqlStatement>
        <sqlStatement name="Q_1">
            …
        </sqlStatement>
    </dataQuery>
    <dataStructure>
        …
    </dataStructure>
</dataTemplate>
```

> **Note**: `xdo_user_name` is a BI Publisher reserved parameter, holding the currently logged-in user. If the report is not passed `P_GL_USER` as a user ID, it uses the upper-case version of the BI Publisher user to set VPD.

Using Utility Packages

Oracle Transportation Management Package Reference

Pre-packaged reports in Oracle Transportation Management leverage a number of PL/SQL packages to simplify query template development. These include:

- utility functions for formatting and logging
- streamlined queries for order, shipment and invoice reports

Table 3 summarizes the available procedures and functions. For more information, please review the `create_rpt` scripts in `<OTM_INSTALL_DIR>\glog\oracle\script8`.

Package	Procedure/Function	Description	Parameters	Returns
vpd	set_user	Sets the user for VPD data security	Oracle Transportation Management user GID	--
	set_user_r	Sets the user and user role for VPD data security	Oracle Transportation Management user GID Oracle Transportation Management user role GID	--
	set_user_fct	Sets the user for VPD data security. Designed for use as the first query in a Data Template	Oracle Transportation Management user GID	true
	set_user_r_fct	Sets the user and user role for VPD data security. Designed for use as the first query in a Data Template	Oracle Transportation Management user GID Oracle Transportation Management user role GID	true
	get_gl_user	Returns the current user for VPD data security	--	Oracle Transportation Management user GID
utc	get_local_date	Converts a UTC timestamp to a location time zone	UTC Timestamp Location GID	Timestamp in the location's time zone
	get_utc_date	Converts a local timestamp to a UTC timestamp based on a location time zone	Local Timestamp Location GID	UTC Timestamp
	get_time_zone	Returns the time zone for a location	Location GID	Time Zone

Package	Procedure/Function	Description	Parameters	Returns
rpt_general	p_insert_log	Logs a record to REPORT_LOG	Unique Filename Report GID Report Job # User Domain up to 3 (name, value) pairs	--
	f_uom_base	Returns the default storage type for a unit of measure	Unit of measure	Default storage type
	f_date_diff	Returns a readable string representing the duration difference of two dates	Ending date Starting date	Duration string
	f_remove_domain	Strips the domain from a GID	GID	XID
	f_format_address	Returns a formatted address for a location	Location GID	Formatted, multi-line address for the location
	f_corporation	Returns the corporation for a location	Location GID	Corporation GID
	f_location_refnum	Returns the value of a specific location reference number	Location GID Reference number qualifier	Reference number value
rpt_order	f_ob_refnum	Returns the value of a specific order base reference number	Order Base GID Reference number qualifier	Reference number value
rpt_ship	f_commodity	Returns the commodity name for a specific item	Item GID	Commodity name
	f_ob_party_location	Returns the location for a specific order base involved party	Order Base GID Involved party qualifier	Location GID

Package	Procedure/Function	Description	Parameters	Returns
	f_or_party_location	Returns the location for a specific order release involved party	Order Release GID Involved party qualifier	Location GID
	f_order_base_gid	Returns the order base associated with an order release	Order Release GID	Order Base GID
	f_party_address	Returns a formatted address for an order release involved party	Order Release GID Involved party qualifier	Formatted, multi-line address for the location
	f_tender_accepted_by	Returns the carrier that accepted a shipment tender	Shipment GID	Service Provider GID
	f_packaging_form_code	Returns the packaging form code for a ship unit specification	Ship Unit Specification GID	Packaging Form Code GID
	f_capacity_rate_offering	Returns the rate offering for a capacity usage	Capacity Usage GID	Rate Offering GID
	f_capacity_time_period	Returns the time period type for a capacity limit	Capacity Limit GID	Time Period Type
	f_lane_source	Returns the source for a lane	XLane GID	Source
	f_lane_destination	Returns the destination for a lane	XLane GID	Destination
	f_equipment_type_name	Returns the name for a equipment type	Equipment Type GID	Equipment Type Name
	f_sellside_cost	Returns the sell-side cost for a shipment	Shipment GID	Sell-side cost
	f_transport_mode_name	Returns the transport mode for a shipment	Shipment GID	Transport Mode

Package	Procedure/Function	Description	Parameters	Returns
	f_get_ship_inv_party_addr	Returns a formatted address for a shipment involved party	Shipment GID Involved Party Qualifier	Formatted, multi-line address for the involved party
	f_get_country_name	Returns the origin or destination country name for a shipment	Shipment GID 'O' for origin, 'D' for destination	Country name
	f_get_pol	Returns the port of lading for a shipment	Shipment GID	Port of lading
rpt_invoice	f_party_location	Returns an involved party location on the invoice	Invoice GID Involved Party Qualifier	Involved party location
	f_party_address	Returns a formatted address for an invoice involved party	Invoice GID Involved Party Qualifier	Formatted, multi-line address for the involved party
rpt_servprov	f_servprov_gid	Returns the Service Provider GID for a particular alias	Service Provider Alias Service Provider Alias Qualifier	Service Provider GID
	f_alias	Returns the Service Provider alias	Service Provider GID Service Provider Alias Qualifier	Service Provider Alias

Table 3: PL/SQL Report Functions

Adding Custom Packages

To add a custom package for use with query templates, the report designer:

- Creates the PL/SQL package specification and the package body definition in the Oracle Transportation Management `reportowner` DB schema
- Compiles both the PL/SQL package specification and body to make sure the code is error free
- Provides grants and permissions so Oracle Transportation Management users can access the custom package. The following scripts are provided under the installation and should be run as `reportowner`.

  `<OTM_INSTALL_DIR>/glog/oracle/script8/reportowner_grants.sql`
  `<OTM_INSTALL_DIR>/glog/oracle/script8/create_public_synonyms.sql`

Dynamic SQL Parameterization

When a report is configured with **Use Report Parameters as Bind Values** selected, parameter values are passed as simple strings to the query template or external system. They can be used as bind variables within queries, whether the report is using a SQL template, data template or some external querying mechanism.

The system, however, supports a more complex specification of parameters. If **Use Report Parameters as Bind Values** is cleared, you are prompted for both a parameter operator and value. E.g., a shipment start time parameter would be entered with an operator like `Before`, `After` or `Between`. The value sent to the query template or external system reflects a dynamic SQL `where` clause. Table 4 lists the possible operators for each type of parameter, the corresponding parameter instruction sent to the report, and dynamic SQL generated for each instruction.

Parameter Type	Criteria	Resulting Condition
Finders	SAME AS~'value'	column='value'
	ONE OF~'value1,value2'	column in ('value1','value2')
	NOT ONE OF~'value1,value2'	column not in ('value1', 'value2')
	BEGINS WITH~'value'	column like 'value%'
	END WITH~'value'	column like '%value'
	CONTAINS~'value'	column like '%value%'
Date/Time	SAME AS~'value'	column = TO_DATE('value', :P_DATE_FORMAT)
	BEFORE~'value'	column < TO_DATE('value', :P_DATE_FORMAT)
	AFTER~'value'	column > TO_DATE('value', :P_DATE_FORMAT)
	BETWEEN~'value1' AND 'value2'	column > TO_DATE('value1', :P_DATE_FORMAT) and column < TO_DATE('value2', :P_DATE_FORMAT)

Parameter Type	Criteria	Resulting Condition
Number	==~value	column=value
	=<~value	column < value
	=>~value	column > value
	=<=~value	column <= value
	=>=~value	column >= value
Any	IS NULL	column is null
	NOT NULL	column is not null

Table 4: Dynamic SQL Parameters

For BI Publisher reports, dynamic SQL is implemented via lexicals. A lexical is a macro expression that evaluates to a PL/SQL variable set in a custom package. Each report supporting dynamic SQL maintains a custom PL/SQL package where:

- The package defines a variable for each parameter, to hold the dynamic SQL.
- The package defines an `AFTERPFORM` function that sets each variable, based on operators and values passed.
- The data template uses the variable as a macro in its where clause. The macro is of the form `&variable name`.

As a simple example, assume you have a report with a single text parameter `P_NAME` that constrains the `s.shipment_name` column in the data template.

The data template would use the lexical to formulate a where clause.

```
<?xml version='1.0' encoding = 'UTF-8'?>
<dataTemplate name = "PKG_SAMPLE_REPORT" defaultPackage="PKG_SAMPLE_REPORT"
version="1.0">

<properties>
   <property name="xml_tag_case" value="upper">
</properties>

<parameters>
   <parameter name="P_GL_USER" dataType="character" defaultValue="DBA.ADMIN"/>
   <parameter name="P_ROLE_ID" dataType="character" defaultValue="ADMIN"/>
    <parameter name="P_NAME" dataType="character" defaultValue="1=1"/>
</parameters>

<dataQuery>
    <sqlStatement name="Q_1">
       <![CDATA[SELECT s.shipment_gid, s.shipment_name
                 FROM shipment s
                     WHERE &P_NAME_PARAM]]>
    </sqlStatement></dataQuery>
```



```
   <dataTrigger name="afterParameterFormTrigger"
source="pkg_sample_report.after_pform"/>

   <dataStructure>
      <element name=”P_NAME_PARAM” dataType=”varchar2”
       value=”PKG_SAMPLE_REPORT.P_NAME_PARAM”>
   </dataStructure>
</dataTemplate>
```

There should be a parameter defined in the query template for receiving the value passed from the UI (in this example it is P_NAME). Also, the global variable defined in the package specification parameter and this parameter should also be of the same name.

The PL/SQL supporting the report could include:

```
Package Specification:
CREATE OR REPLACE PACKAGE PKG_SAMPLE_REPORT IS

    P_NAME VARCHAR2(32766) := '1=1';
    P_NAME_PARAM VARCHAR2(32766) := ‘1=1’;
   P_GL_USER VARCHAR2(128);

   FUNCTION AFTERPFORM RETURN BOOLEAN;
END PKG_SAMPLE_REPORT;

Package Body:
CREATE OR REPLACE PACKAGE BODY PKG_SAMPLE_REPORT IS

    FUNCTION AFTERPFORM RETURN BOOLEAN IS
       P_NAME_PARAM := s.shipment_name ||
          REPORTS_LIBRARY.GET_FILTER_CONDITION(P_NAME, NULL, ‘Y’);
    END AFTERPFORM;

END PKG_SAMPLE_REPORT;
```

This custom package name is linked to the report’s data template using the defaultPackage attribute.

When the query template starts executing, it sets the values for all the global variables defined in the specification of the package (defined for the query template). The BI Publisher engine looks for the global variables with the same name as defined in the query template.

Using a Saved Query as a Report Input Parameter

Advantages

While defining a report, you can give as many different input parameters as required to constrain the data to be fetched. However, if there are too many parameters, it becomes inconvenient to enter a great number of values before running the report.

The Saved Query approach enables you to save different constraints for an object. A saved query is nothing but an SQL statement derived using an Oracle Transportation Management finder for an object, where different fields can be defined for filtering the data. All these constraints are saved as “where” clauses in the query, and the select statement is the primary key of the object.

You define constraints using an Oracle Transportation Management finder, and save these constraints as a saved query. When you run the report, you use the saved query to provide the input parameters. The saved query is then used to filter the output on the report.

How to Use the Saved Query as Report Input Parameter

The idea here is to use the query in the SQL_FIND_ALL column of the SAVED_QUERY table, for a particular SAVED_QUERY_GID, as a sub query for the main query defined in the query template. We can use this sub query to filter the objects which are returned by the saved query.

Follow the below steps to configure a Saved Query as an input parameter to any report. The example shown below is a Bill Of Lading report.

Report Definition

1. Define the report input parameter:
   - **Parameter Type**: *Saved Query*
   - **Data Query Type**: *SHIPMENT* (this is the primary key of the object on which the saved query has been defined). (Assume the report parameter name defined here is P_L_SHIP_SAVED_QUERY).
2. Save the report definition with this saved query as one of the parameter. Defining a saved query as the parameter means that the user is providing all the constraints he wants to specify, on a specific object that exists in the saved query. Depending on the need, other parameters can also be included.
3. Navigate to the Report Input Parameter screen for the above defined report. There the saved query parameter appears in the Saved Query drop down for the object pointed to by the Data Query Type (in this example, SHIPMENT).
4. Run the report using the saved query.

Parameter Creation in Query Template Creation

There should be a parameter defined in the query template for receiving the value passed from the UI. The parameter name should be the same as what we have defined in the report definition.

```
<parameter>
    <parameter name="P_GL_USER" dataType="character” defaultValue="DBA.ADMIN"/>
    <parameter name="P_ROLE_ID" dataType="character" defaultValue="ADMIN"/>
    <parameter name="P_L_SHIP_SAVED_QUERY" dataType="character" />
</parameters>
```

Since a dynamic SQL has to be formed using this saved query, you will have to create a custom package and define it in the query template attached to the report. This custom package is defined using the defaultPackage attribute in the query template as follows:

```
<dataTemplate name="bill_of_lading" defaultPackage=" bill_of_lading"
version="1.0">
```

Package Specification

In the package defined for the query template, you need to create two global variables, one with the same name as we have in the data query template (`P_L_SHIP_SAVED_QUERY`), and another for holding the dynamic SQL (`P_L_SHIPMENT_ID_PARAM`).

```
P_L_SHIPMENT_ID1 VARCHAR2(32766) := '1=1';
P_L_SHIPMENT_ID_PARAM VARCHAR2(32766) := '1=1';
P_L_SHIP_SAVED_QUERY_ID VARCHAR2(32766) := '1=1';
```


The parameter defined in the query template and the global variable in the package specification should be the same (P_L_SHIP_SAVED_QUERY_ID).

When the query template starts executing, it sets the values for all these global variables defined in the specification of the package (defined for the query template). The BI Publisher engine looks for the global variables with the same name as defined in the query template.

Data triggers

Once the global variables are set, the dataTriggers defined in the query template start getting executed. In Oracle Transportation Management, it is advisable to use the afterpform (after parameter formation) function in order to manipulate the parameters. The data triggers are defined as follows:

```
<dataTrigger name="afterParameterFormTrigger"
source="bill_of_lading.afterpform"/>
<dataTrigger name="beforeReportTrigger" source="bill_of_lading.beforereport"/>
```

The dynamic SQL can be formed in either the afterpform function or the set_lexical_parameter function, which is called from afterpform function. For example, let us say the column is "shipment_gid" and the saved query is for "shipment" as is the data query type. `P_L_SHIP_SAVED_QUERY_ID` is the parameter name for saved query. `P_L_SHIPMENT_ID_PARAM` is the variable that shall hold the dynamic SQL.

The dynamic query can be formed as follows:

```
PROCEDURE SET_LEXICAL_PARAMETERS IS
subQuery varchar2(4000);
v_sql varchar2(1000);

 BEGIN

      v_sql:='select sql_find_all from saved_query where saved_query_gid =
     P_L_SHIP_SAVED_QUERY_ID';

     execute immediate v_sql into subQuery using P_L_SHIP_SAVED_QUERY_ID;

     IF VALUE_ENTERED (P_L_SHIP_SAVED_QUERY_ID) THEN
     P_L_SHIPMENT_ID1 := 'BOL_HEADER_BOV.SHIPMENT_GID in (' || subQuery || ')';
     P_L_SHIPMENT_ID_PARAM := 'SHIPMENT_GID in (' || subQuery || ')';

     ELSE
      P_L_SHIPMENT_ID_PARAM := '1=1';
      P_L_SHIPMENT_ID1 := '1=1';
     END IF;

     END SET_LEXICAL_PARAMETERS;
```

You can now use the variable that holds the dynamic sql in the query template.

In order to use the dynamic sql where clause, it has to be defined as an element in the data structure of the query template as follows:

```
<dataStructure>
 <element name="P_L_SHIPMENT_ID1" dataType="varchar2"
value="BILL_OF_LADING.P_L_SHIPMENT_ID1"/>
  <element name="P_L_SHIPMENT_ID_PARAM" dataType="varchar2"
value="BILL_OF_LADING.P_L_SHIPMENT_ID_PARAM"/>
```


This element can then be used as a lexical parameter (precede the name of the element with the "&" symbol) in the query, which gets substituted as it is in the query.

```
<sqlStatement name="Q_1">
          <![CDATA[SELECT ALL
STOPS_INFO_BOV.SHIPMENT_GID,
STOPS_INFO_BOV.STOP_NUM,
STOPS_INFO_BOV.CONTACT_NAME,
STOPS_INFO_BOV.PHONE,
STOPS_INFO_BOV.PARTY_NAME,
STOPS_INFO_BOV.ADDRESS1,
STOPS_INFO_BOV.ADDRESS2,
STOPS_INFO_BOV.ADDRESS3,
STOPS_INFO_BOV.UNIT,
STOPS_INFO_BOV.ARRIVAL_DATE,
STOPS_INFO_BOV.ARRIVAL_TIME,
STOPS_INFO_BOV.DEPARTURE_DATE, STOPS_INFO_BOV.DEPARTURE_TIME,
STOPS_INFO_BOV.CARRIER_NAME

FROM REPORTOWNER.STOPS_INFO_BOV
WHERE &P_L_SHIPMENT_ID_PARAM]]>
</sqlStatement>
```

On execution the 'where' clause in the above query becomes:
WHERE SHIPMENT_GID IN (subquery)

bill_of_lading_query_template.xml

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<dataTemplate name="bill_of_lading" defaultPackage="bill_of_lading"
version="1.0">
   <properties>
      <property name="xml_tag_case" value="upper"/>
   </properties>
   <parameters>
      <parameter name="P_GL_USER" dataType="character"
defaultValue="DBA.ADMIN"/>
      <parameter name="P_ROLE_ID" dataType="character" defaultValue="ADMIN"/>
      <parameter name="P_L_SHIP_SAVED_QUERY" dataType="character"/>
   </parameters>
   <lexicals/>
   <dataQuery>
      <sqlStatement name="Q_1">
         <![CDATA[SELECT ALL
     STOPS_INFO_BOV.SHIPMENT_GID,
     STOPS_INFO_BOV.STOP_NUM,
     STOPS_INFO_BOV.CONTACT_NAME,
     STOPS_INFO_BOV.PHONE,
     STOPS_INFO_BOV.PARTY_NAME,
     STOPS_INFO_BOV.ADDRESS1,
     STOPS_INFO_BOV.ADDRESS2,
     STOPS_INFO_BOV.ADDRESS3,
     STOPS_INFO_BOV.UNIT,
     STOPS_INFO_BOV.ARRIVAL_DATE,
     STOPS_INFO_BOV.ARRIVAL_TIME,
     STOPS_INFO_BOV.DEPARTURE_DATE,
     STOPS_INFO_BOV.DEPARTURE_TIME,
     STOPS_INFO_BOV.CARRIER_NAME
```



```
     FROM REPORTOWNER.STOPS_INFO_BOV

                          WHERE &P_L_SHIPMENT_ID_PARAM]]>
      </sqlStatement>
      <sqlStatement name="Q_2"></sqlStatement>
      <sqlStatement name="Q_3"></sqlStatement>
      .
          .
          .<!--SQL Queries-->
      .
      .

   </dataQuery>
   <dataTrigger name="afterParameterFormTrigger"
source="bill_of_lading.afterpform"/>
   <dataTrigger name="beforeReportTrigger"
source="bill_of_lading.beforereport"/>
   <dataStructure>
        <element name="P_L_SHIPMENT_ID1" dataType="varchar2"
value="BILL_OF_LADING.P_L_SHIPMENT_ID1"/>
        <element name="P_L_SHIPMENT_ID_PARAM" dataType="varchar2"
value="BILL_OF_LADING.P_L_SHIPMENT_ID_PARAM"/>
     .
     .
     .
     . <!-- Define elements and groups as per your queries and grouping
requirements -->
     .
     .
     .
   </dataStructure>
   <dataTrigger name="afterReportTrigger"
source="bill_of_lading.afterreport()"/>
</dataTemplate>
```

Note: Report designers should avoid dynamic SQL when creating reports. The additional PL/SQL complexity and data template complexity outweighs the advantages of a dynamic Where clause. For example, using a dynamic condition to control a date range can be easily replaced with two parameters: one defining the earliest date; and one the latest date.

SQL Parameterization is presented here to aid in migration from Oracle Reports. Reports designed for previous versions of Oracle Transportation Management rely on dynamic where clauses and will be migrated with lexicals. For Oracle Transportation Management version 6.2, most reports shipped with the system were migrated from Oracle Reports and support dynamic SQL.

5. Advanced Layout

Navigating Oracle Transportation Management Integration XML

When using an Integration template for data generation, a report designer needs a good understanding of XPath. Oracle Transportation Management Integration XML is based on the following schema:

```
<OTM_INSTALL_DIR>/glog/config/GLogXML.xsd
```

The Integration template can be configured to send BI Publisher the overall `<Transmission>` element or the specific business object element (i.e. one of the elements contained by `<GLogXMLElement>`). In either case, navigating to data generally requires a more sophisticated XPath expression than the BI Publisher Microsoft Word add-in generates.

As an example, assume a shipment report is based on integration XML. To display the source location, a designer retrieves the Location XID for the first shipment stop. The shortest XPath unique expression would be:

```
ShipmentStop[StopSequence=1]/LocationRef/LocationGid/Gid/Xid
```

To specify this in Microsoft Word, the designer opens up the Advanced tab for the edit field and specifies the XPath within a BI Publisher markup:

```
<?ShipmentStop[StopSequence=1]/LocationRef/LocationGid/Gid/Xid?>
```

The reuse of elements within the schema forces more complex field specification.

Namespaces

By default, integration XML is provided without namespace qualifiers. This simplifies XPath specification but may not support all possible schema elements. Global Trade Management objects, for example, may include two elements from two namespaces: the default namespace for Oracle Transportation Management; and one for elements specific to Global Trade Management.

If namespaces are enabled, a report designer must register any namespace prefixes in his layout and use the prefixes in XPath expressions. For example, assume the XML has the default namespace:

```
<xsl:schema xmlns=http://xmlns.oracle.com/apps/otm ...>
```

The report designer must add the following markup prior to laying out data fields:

```
<?namespace:x=http://xmlns.oracle.com/apps/otm?>
```

where `x` is some prefix to be used for the associated XPath elements. To specify this in Microsoft Word, the designer must add a text form field to Microsoft Word for which help text holds the markup. Once the namespace is registered, XPath markup for the source location would be:

```
<?x:ShipmentStop[x:StopSequence=1]/x:LocationRef/x:LocationGid/x:Gid/x:Xid?>
```


Properties

Table 5 lists properties to control the format of integration XML provided to BI Publisher:

Property	Type	Values	Description	Default Value
glog.bipreports.integration.formatOutboundXML	boolean	true/false	If true, integration XML for reports is formatted for readibility	false
glog.bipreports.integration.transmissionXML	boolean	true/false	If true, the full Transmission XML is output. This is useful for customers basing reports on the full Oracle Transportation Management schema. If false, only the specific GLogElement is included.	false
glog.bipreports.integration.stripInstruction	boolean	true/false	If true, the <?xml> instruction is stripped from the output	false
glog.bipreports.integration.stripNamespace	boolean	true/false	If true, the Oracle Transportation Management namespace is stripped from the output.	true

Table 5: Integration XML Properties

PDF Customization

A report designer can customize the format template to support three features specific to PDF report output:

- **Text Watermarks**: You can control the watermark's font, font size, angle, color, and bottom left coordinates (as measured in pixels from the bottom left of the screen).
- **Image Watermarks**: You can upload an image for the watermark and control its lower left and upper right coordinates. The watermark is scaled to fit the specified rectangle.
- **Page Numbers**: You can control the starting page number, font, font size, and bottom left coordinates (as measured in pixels from the bottom left of the screen).

If you pick an output type other than PDF, these settings are ignored.

XSL vs. XSL:FO Transformation

Embedded reports are designed around the BI Publisher Transformation API. A common use case of this API is to transform XSL:FO layouts with XML data to generate report output. BI Publisher supports a number of XSL/XML transformation outputs including: HTML, PDF, RTF, Excel, CSV, and eText.

Oracle Transportation Management, though, supports a more direct transformation. By uploading a standard XSLT stylesheet format to the format template, a designer can avoid BI Publisher. Instead, the system uses Xalan to transform the XSLT with the XML into some structured output. This is controlled by instructions in the XSLT but typically generates HTML. To use the Xalan transformation, upload the stylesheet, but be sure to leave the `XSL-FO Template` check box unchecked.

6. Scalability

Tier Control

The following are steps in the generation of an embedded report:

1. Template Retrieval
2. Data Generation
3. Transformation
4. Distribution (e.g. e-mail, IPP)

For scheduled and automated reports, these steps must run on the application tier. For ad-hoc reports, though, properties control which steps run on which tier. Running some or all of these steps on the web tier can increase performance.

As an example, assume a report has a small template, generates little data but produces large report content. Transforming the report on the application tier may be infeasible because the returned content exceeds RMI buffers. Running steps 1 and 2 on the application tier, but step 3 on the web tier avoids RMI communication limits. If the web server is behind a firewall and has access to the database, running steps 1 and 2 on the web tier could further optimize report generation.

By default, all four steps are run on the application tier. This allows the web tier to reside outside the firewall without access to the database. It also allows application clustering to scale computationally intensive reports.

The following properties specify tier control for ad-hoc reports:

Property	Description	Default	Comments
glog.bipreports.appTier.transform	Runs steps 1-4 on the application tier	true	if false, the web tier must have database access
glog.bipreports.appTier.query	Runs steps 1 and 2 on the application tier	true	if **transform** is false and **query** is false, transformation is run on the web tier
glog.bipreports.appTier.distribute	Runs step 4 on the application tier	true	if **transform** is false and **distribute** is false, distribution (e-mail and IPP) is run on the web tier

If ad-hoc reports are generated on the application tier, but the web tier and application tier reside on the same physical server and can access a shared directory, the bandwidth to return large report content via RMI can be avoided. When the property `glog.bipreports.report.share.browser` is true, the report is saved to the `glog.bipreports.report.path` directory. When control is returned to the web tier, it reads the report file to output content to the browser.

Application Server Scalability

Report processing can be scaled on the application tier by assigning `Reports` functionality to a cluster of application servers. All report generation is delegated to the cluster, whether ad hoc, scheduled, or automated.

If the cluster is dedicated only to `Reports`, the cluster overhead is minimal. A reports cluster does not require JMS synchronization or business object locking. The only coordination it needs with other servers is for Process Control handling.

Web Server Scalability

If the web tier is used for ad-hoc report generation (i.e. `glog.bipreports.appTier.transform=false`), report processing can also be scaled on the web tier by:

- enabling Web Scalability
- defining web servers and web clusters
- delegating `Reports` functionality to a dedicated web cluster

Please see the Application Scalability Guide for more information.

External BIP Server Farms

Though application server scalability can be used to isolate and scale BI Publisher report generation, report designers also have the option to generate reports on a set of dedicated BI Publisher servers. From the standpoint of Oracle Transportation Management, a report running on a BI Publisher server farm is an external report with an `Embedded` content type. Requests to generate the report are sent to one of the BI Publisher servers, which returns content to Oracle Transportation Management.

BI Publisher provides a web service to invoke reports. Oracle Transportation Management provides a special servlet, `glog.webserver.report.BIPReportExternalRequestServlet`, to map a third party report URL to the web service call. Table 6 summarizes the request parameters needed by the servlet to properly direct BI Publisher.

Request Parameter	Description	Required	Notes
BIP_HOST	Host name running the BI Publisher server	Yes	
BIP_PORT	Port for BI Publisher connections	No	defaults to 9705
BIP_USER	BI Publisher user	Yes	should match an Oracle Transportation Management user to ensure proper data security
BIP_PASSWORD	Bi Publisher login password	Yes	
BIP_REPORT_PATH	A path to the report	Yes.	spaces must be encoded

Table 6: External BI Publisher Parameters

As an example, assume BI Publisher Enterprise is running on a host name `bip-host`. A report `Simple Commercial Invoice` is available in BI Publisher, in the `OTM Reports` folder. The report needs to be run in the `CUSTOMER` domain and a BI Publisher user `customer.admin` is available with a password of `CHANGEME`. Finally, the Oracle Transportation Management web server host is `otm-web`. The third party URL to use in Oracle Transportation Management is:

http://otm-web/GC3/glog.webserver.report.BIPReportExternalRequestServlet?BIP_HOST=bip-host&BIP_USER=customer.admin&BIP_PASSWORD=CHANGEME&BIP_REPORT_PATH=/OTM%20Reports/Simple%20Commercial%20Invoice/Simple%20Commercial%20Invoice.xdo

When we define a report in OTM and try to access the report while generating the data content, the VPD is always applied.

If the reports are created on an 11g or 10g server and accessed from OTM through web services, the VPD will not be applied. In order to have the VPD applied for these reports newly created outside of OTM, you will have to explicitly set the VPD using the before data event triggers which can call a PLSQL function and can set the VPD accordingly.

For example:

1. If you are using 10g reports server and using a data template for creation of the report content, then use the dataTrigger tag for specifying the PLSQL function to call the VPD:

   ```
   <dataTrigger name="afterParameterFormTrigger"
   source="reports_library.set_vpd(:P_GL_USER, :P_ROLE_ID)"/>
   ```

2. If you are using an 11g report server, call the PLSQL function using the Event Triggers which are called before data.

7. Troubleshooting

When generating a report, errors can occur in the BI Publisher layer when generating data or transforming the data against a stylesheet. If fatal, an exception is thrown to Oracle Transportation Management. For an ad-hoc report request, the exception is shown to the user; for scheduled or automated reports, it is written to the exception log.

Report Logging

Two types of logging are available to diagnose report issues:

- Oracle Transportation Management logging. The `REPORT` log ID outputs each major step of report creation (template retrieval, data generation, transformation, and distribution). The logging may be written to the web log or various application logs, depending on tier control settings. The `REPORT_DETAILS` log ID provides more detailed information on the transformation engine.
- BI Publisher logging. The BI Publisher engine maintains a separate logging subsystem to track data generation and transformation progress. The following properties control where and what BI Publisher will log:

Property	Description	Default
glog.bipreports.log.filename	The file to hold BI Publisher logging	bipublisher.log
glog.bipreports.log.level	The level of detail for logging. Valid values are: 0 = none 1 = all statements 2 = all procedures 3 = all events 4 = exceptions only 5 = errors only 6 = unexpected errors only	4
glog.bipreports.log.maxsize	Maximum size of the log file before spooling to a backup	1000000
glog.bipreports.log.numBackups	Maximum number of log file backups	5

Note: The BI Publisher log level also controls exception output. While generating a report, Oracle Transportation Management captures any BI Publisher logging[11]. If an exception occurs, this logging is prepended to the exception stack trace. Given a reporting error, detailed BI Publisher logging can be turned on via the properties servlet. When the user tries to rerun the report, the exception trace will include the BI Publisher logging and aid in troubleshooting.

[11] Currently, BI Publisher does not provide a mechanism to capture logs for a particular request. The log output may show results from a number of synchronous report requests.

Intermediate File Persistence

During report generation, content and report files are written to the disk before being distributed. By default, these files have a short lifespan. Once the report is generated, the content file is deleted. Once the report is distributed, the report file is deleted. For troubleshooting, it may be useful to view these files. They can be persisted via the following properties:

Property	Description	Default
glog.bipreports.content.persist	If true, content files are not removed	false
glog.bipreports.report.persist.browser	If true, report files used for browser views are not removed	false
glog.bipreports.report.persist.attachment	If true, report files used for e-mails attachments are not removed	false
glog.bipreports.report.persist.printer	If true, report files used for IPP printing are not removed	false

8. Oracle Reports Migration

Prior to Oracle Transportation Management 6.2, reports were generated with Oracle Reports. There are three options to leverage these reports in 6.2:

- Redefine the reports as a third party report. Each report will require a URL to the Oracle Reports Server, specifying the specific report to be run.
- Migrate the reports to BI Publisher, maintaining dynamic SQL criteria. This is the default behavior of the BI Publisher migration tools. It requires the maintenance of a PL/SQL package per report and lexicals within the data template.
- Migrate the reports to BI Publisher, removing support for dynamic SQL. This would require using the BI Publisher migration tools to maintain the layout and data queries. Once migrated, though, the report designer would remove the PL/SQL package and modify the data template to use simple bind parameters.

Examples of these migrations will be available in future versions of this document.

9. Additional Resources

Oracle Business Intelligence Suite Enterprise Edition Documentation Library:

http://download.oracle.com/docs/cd/E10415_01/doc/nav/portal_booklist.htm

A BI Publisher developer's diary...

http://blogs.oracle.com/BIDeveloper/

BI Publisher Forum

http://forums.oracle.com/forums/forum.jspa?forumID=245