Oracle® Agile Product Lifecycle Management for Process

Custom Report Configuration Guide

Extensibility Pack 2.7

E37243-01

September 2012

ORACLE®

Copyrights and Trademarks

Agile Product Lifecycle Management for Process

Contents

Custom Report Configuration

Purpose

This guide describes how to configure custom reports for the Oracle Agile Product Lifecycle Management (PLM) for Process Reporting module.

Overview

The Reporting application allows client to organize, configure, secure, and launch custom reports. The configuration is managed in the `CustomerReportExtensions.xml` located in the `\Config\Extensions\` directory. Clients can configure custom reports, specify the categorization of the reports, configure visibility rules via custom classes, and define the various report parameters to display.

Reports are categorized by two grouping levels: Report Contexts and Report Groups, each of which can be secured by configuring security classes.

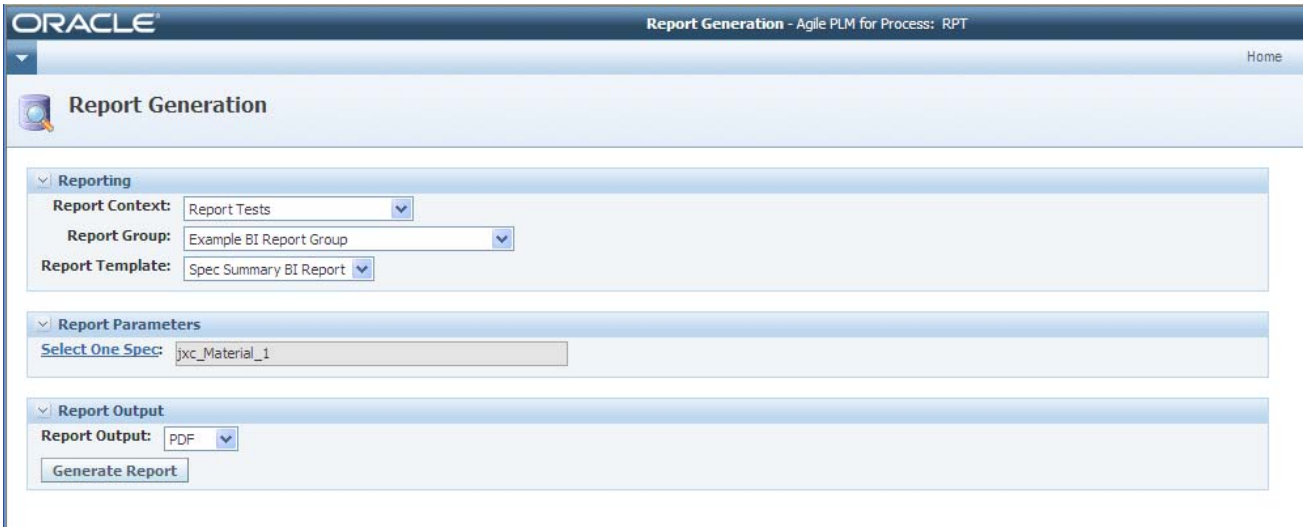
Report parameters can use existing pop-ups found throughout the application, or use custom-defined parameters. The parameter values are then sent to a reporting service, such as Oracle BI Publisher or SQL Server Reporting Services, which process the parameters to make custom SQL queries to produce the report output. The reporting service then returns the results to the user.

For details on configuring the reporting service, see the *Oracle Agile Product Lifecycle Management for Process Configuration Guide*.

Report Generation Screen

The report generation screen allows users to select an available report, enter report parameters, select an output format, and launch the report.

Figure 1. Sample Report Generation Screen



The Report Generation screen is comprised of three main sections:

1. Reporting--Provides a way to organize, categorize, and secure reports.
2. Report Parameters—Presents parameter input and options for each report.
3. Report Output—Presents the possible reporting output formats, such as XML and PDF.

The data that is displayed to users is driven from the `CustomerReportExtensions.xml` file. This guide explains how to configure this file for custom reporting needs.

Configuring the CustomReportExtensions.xml File

Customizing the report configuration primarily consists of defining any new report parameters types that will be available to the reports, and configuring the custom reports. Any changes to the CustomerReportExtensions.xml file will require the ProdikaReporting web application to be restarted.

File Structure Overview

Common objects, such as report parameter input types and common reports are defined in the ServiceConfig section. The reports, their categorization, security, and parameter details are defined in the ReportContexts section. The Contextual Reporting section defines ways to launch specific reports from the various web applications (such as GSM, SCRM) based on several criteria.

A high-level outline of the Report Contexts and ServiceConfig hierarchies found in the `CustomerReportExtensions.xml` file follows:

```
ServiceConfig
   o  ParameterTypes
         ▪  ParameterType
   o  CommonReports
ReportContexts
   o   ReportContext
```

- ContextCriteria
- ReportGroup
  - Parameters
  - Reports
    - Report
      - AllowedOutputTypes

Reports are categorized by two grouping levels: Report Contexts and Report Groups.

Report Contexts represent the highest level of report organization, and contains multiple report groups. These are displayed in the Report Context drop-down list.

Report Groups provide a way to group multiple reports that all use the same report parameters. The parameters types specified for each ReportGroup are defined in a separate section called ServiceConfig.

The following sections contain detailed information on how to configure each of these nodes.

Configuring Report Parameter Types

All report parameter types that are used by the reports must be defined in the `\ServiceConfig\ParameterTypes` section. These parameters may then be referenced by the reports defined in the `\ReportContexts` section. Several predefined report parameters are available for use, but you may also create their own custom parameter types.

There are three different types of report parameters:

1. Simple type parameters – Parameter input types, such as a string or date field input
2. DataSource parameter types – Classes that are responsible for retrieving data for display
3. Models – Predefined Oracle Agile PLM data retrievers that are already used in the application

Each parameter type must specify a user interface element that will be used to display the report parameter to the user. DataSource and Model report parameter types must also specify the method used to retrieve the data to display.

Global Report Parameters

The following report parameters are automatically passed in and available to each report server request:

1. `paramCurrentUser` – The current user's PKID (40 character unique identifier string)
2. `paramCurrentLanguage` – The current user's language ID setting (see SupportedLanguages table)

Note: Using DataSources and Models requires detailed technical knowledge of the Oracle Agile PLM for Process application and may require writing custom classes. See *Appendix A – Creating Custom Report Parameter Types* for details.

<ParameterTypes> Node

The ParameterTypes node contains a list of ParameterType nodes. Each ParameterType defines an input parameter that may then be used to display selection criteria for reports.

Attributes

Attribute Name	Description	Required?
N/A		

Child Nodes

<ParameterType>

Parent Node

<ServiceConfig>

<ParameterType> Node

The ParameterType node defines an input parameter that may then be used to display selection criteria for reports. Each Parameter Type must also declare the user interface control to use for displaying the report parameter to the user. The child nodes required are based on the specified web control.

Attributes

Attribute Name	Description	Required?
type	Unique identifier for this parameter type. This value will be used to refer to the parameter type when configuring which parameters are used for each report in a Report Group.	Yes

Attribute Name	Description	Required?
webControl	The name of one of the following available web controls: ReportingControls/**StringInput**.ascx – An input box that allows alphanumeric entry ReportingControls/**DateInput**.ascx – A date picker control preset to the current date ReportingControls/**MonthYearInput**.ascx – A month selection control and a year input control ReportingControls/**LookupInputSingleSelect**.ascx – An drop-down input allowing a single selection from data specified by a DataSource ReportingControls/**LookupInputMultiSelect**.ascx – A pop up control allowing multiple selections from data specified by a DataSource ReportingControls/**TreeViewInput**.ascx – A pop up control for nested hierarchies, allowing a single or multiple selections from data specified by a DataSource ReportingControls/**EQTInput**.ascx – A pop up control with differing behavior based on which existing EQT View is used See *Appendix A – Creating Custom Report Parameter Types* for more details on how each web control functions.	Yes

Child Nodes

The possible child nodes will depend on which web control is used:

StringInput, DateInput: No child controls

MonthYearInput : Allows for customizing the parameter names for the month and year controls

- <MonthParamNameExtension> – InnerText should be set to the name extension
- <YearParamNameExtension> – InnerText should be set to the name extension

LookupInputSingleSelect, LookupInputMultiSelect:

- <DataSource>

TreeViewInput:

- <DataSource>
- <MultiSelect> – InnerText should be 'Yes' to enable selection of multiple entries, or 'No' for single selection.

EQTInput:

- <Model>

Parent Node

<ParameterTypes>

<DataSource> Node

The DataSource element is used to specify a class that will retrieve data to be used in the given web control. Existing DataSources may be used to retrieve data, or custom DataSource classes may be created to fulfill more specific requirements.

Attributes

Attribute Name	Description	Required?
className	The full path of the DataSource class to use, along with any parameters. Format should be in the form of an Object Loader URL. Please see *Appendix C – Object Loader URLs* for details. Example: `Class:Xeno.Web.UI.Controls.DataSources.LookupServiceListViewDataSource,XenoWebControls$Allergens|Allergens`	Yes

Child Nodes

N/A

Parent Node

<ParameterType>

<Model> Node

The Model element is used to specify an existing search model (EQTModel) that will retrieve data to be used in the given EQTInput web control. The searches available through the EQT models may be much more detailed and allow the user to specify exact search criteria.

Attributes

Attribute Name	Description	Required?
displayVariableIndex	Indicates which return value from the EQT popup selection should be displayed. When an EQT view returns a value for display, the /DisplayColumns/ColumnInfo element (from the Model node) includes an attribute `provideInSelectJS="true"`. If multiple display columns in the EQT Model have the provideInSelectJS="true" attribute, the `displayVariableIndex` attribute determines which one should be displayed to the user. For Models that only return one display value, the displayVariableIndex attribute should be set to "1".	Yes

InnerText Value

The path to the EQT Model. See *Appendix A – Creating Custom Report Parameter Types* for more details.

Child Nodes

N/A

Parent Node

<ParameterType>

Configuring Reports

<ReportContext> Node

The ReportContext node is used to group and secure related Report Groups together. All available Report Contexts are displayed to the user in the Report Context drop-down list of the Report Generation Screen.

Attributes

Attribute Name	Description	Required?
Name	The name of the report context that will be displayed to the user in the Report Generation screen.	Yes
SecurityFactoryRef	The security plug-in to use to limit access to this Report Context. See *Appendix B – Applying Security to Reports* for more details.	No

Child Nodes

<ReportGroup>
<ContextCriteria>

Parent Node

<ReportContexts>

<ContextCriteria> Node

This node is currently not used.

Child Nodes

<Parameters>

Parent Node

<ReportContext>

<ReportGroup> Node

The ReportGroup node is used to group and secure related Report Groups together. All available Report Contexts are displayed to the user in the Report Context drop-down list of the Report Generation Screen.

Attributes

Attribute Name	Description	Required?
name	Used to uniquely identify this report group in the configuration files.	Yes
label	The display name that will appear to users or the translation lookup key. See isLabelTranslatable below.	Yes
isLabelTranslatable	**true** – Will use the value of the label attribute to look up the report group name in the translation caches, using the translationCache attribute, and return the translation. **false** – Will use the value of the label attribute as the display name of the report group.	No
translationCache	The name of the translation cache used to look up the translation for the label value, if the isLabelTranslatable attribute is set to true.	Yes if isLabelTranslatable is true
securityFactoryRef	The security plugin to use to limit access to this Report Context. The value must be a valid Object Loader URL. See *Appendix B – Applying Security to Reports* for more details.	No
serviceLocation	If present, will use the given value as the web service location to use to process all reports in this report group; otherwise, will use the serviceLocation attribute value in the\ServiceConfig\ReportingService element.	No

Attribute Name	Description	Required?
reportEngineName	If present this will specify which report engine all reports within that group will use. The options are: SQLReportingService OracleBIPublisher10_1_3_3 OracleBIPublisher10_1_3_4 OracleBIPublisher11_1_1_5 Refer to the Install/Upgrade Guide for setting the report engine configuration entries. If no report engine is specified it will use the default report engine: SQLReportingService Note: the default report engine can be changed in the EnvironmentSettings.config file, by setting the default="true" attribute to the desired reportEngine entry in the /ProdikaReporting/ServiceConfig child nodes.	No

Child Nodes

- Parameters

  Reports

Parent Node

<ReportContext>

<Parameters> Node

The Parameters node contains a list of Parameter nodes. Each Parameter will be used as input criteria for the reports in the current ReportGroup.

Attributes

Attribute Name	Description	Required?
N/A		

Child Nodes

Parameter

Parent Node

<ReportGroup>

<Parameter> Node

The Parameter node will be used as an input criterion for the reports in the current ReportGroup. The parameter declared here uses a Parameter Type that must be defined in the ServiceConfig section of the configuration.

Attributes

Attribute Name	Description	Required?
name	The name of the parameter that will be passed to the Report Server. Must be unique within the <Parameters> node.	Yes
label	The display name that will appear to users or the translation lookup key. See isLabelTranslatable below.	Yes
isLabelTranslatable	**true** – Will use the value of the label attribute to look up the parameter name in the translation cache of the parent ReportGroup and return the translation. **false** – Will use the value of the label attribute as the display name of the parameter.	No – Will use false if not specified
type	The name of the ParameterType from the \ServiceConfig\ParameterTypes list.	Yes
Required	**true** – This parameter will be a required criterion for launching the report. **false** – This parameter will be an optional criterion for launching the report.	No – Will use false if not specified

Child Nodes

N/A

Parent Node

<Parameters>

<Reports> Node

The Reports node contains a list of Report elements. Each Report will use the same Report Parameters as input criteria in the current ReportGroup.

Attributes

Attribute Name	Description	Required?
N/A		

Child Nodes

<Report>

Parent Node

<ReportGroup>

<Report> Node

The Report node represents an individual report that can be launched using the report parameters defined by the current report group. Reports can reference previously declared common reports and use the attribute values defined for that original report.

Attributes

Attribute Name	Description	Required?
id	Used to uniquely identify reports that can be reused throughout the reporting configuration. If any other report in the configuration matches the ***id*** value, an error is thrown. See the *Configuring Common Reports* section on page 17.	No
idref	May be used to reference a previously declared Common Report entry. The ***idref*** value must match the unique ***id*** value of the desired report. If a matching reference report is found, no other attributes from this element are used. If no matching report is found, an error is thrown. See the *Configuring Common Reports* section on page 17.	No
name	Used to uniquely identify this report in the configuration files within the report group.	Yes, unless ***id*** attribute is used
label	The display name that will appear to users or the translation lookup key. See isLabelTranslatable below.	Yes
isLabelTranslatable	**true** – Will use the value of the label attribute to look up the report name in the translation cache of the parent ReportGroup and return the translation. **false** – Will use the value of the label attribute as the display name of the report.	No – Will use false if not specified
reportPath	Path to the individual report on the reporting server.	Yes
serviceLocation	If present, will use the given value as the web service location to use to process this report; otherwise, will use the ReportGroup's value.	No
translationCache	If present, will use the given value as the name of the translation cache to use to process this report's label; otherwise, will use the ReportGroup's value.	No

Attribute Name	Description	Required?
SecurityFactoryRef	The security plug-in to use to limit access to this report. The value must be a valid Object Loader URL. See *Appendix B – Applying Security to* Reports and *Appendix C – Object Loader URLs* for more details.	No
template	The name of the BI Publisher layout template to use for the report output.	Yes if using BI Publisher for report generation; No otherwise
reportEngineName	If present this will specify which report engine this report will use. The options are: SQLReportingService OracleBIPublisher10_1_3_3 OracleBIPublisher10_1_3_4 OracleBIPublisher11_1_1_5 Refer to the Install/Upgrade Guide for setting the report engine configuration entries. If no report engine is specified it will use the default report engine: SQLReportingService Note: the default report engine can be changed in the EnvironmentSettings.config file, by setting the default="true" attribute to the desired reportEngine entry in the /ProdikaReporting/ServiceConfig child nodes. #	No

Child Nodes

<AllowedOutputTypes>

Parent Node

<Reports>

<AllowedOutputTypes> Node

The AllowedOutputTypes node contains a list of OutputType nodes indicating the possible output formats available for the report. Individual reports therefore restrict the full list of allowed output types defined in the \ServiceConfig\ActiveReportOutputTypes node.

Attributes

Attribute Name	Description	Required?
N/A		

Child Nodes

AllowedOutputType

Parent Node

<Report>

<AllowedOutputType> Node

The AllowedOutputType node indicates a possible output format available for the given report. Entries are limited to values defined in the reporting framework's list, such as EXCEL, PDF.

Attributes

Attribute Name	Description	Required?
key	Unique identifier for the output type	Yes
value	The display name	Yes
extension	File extension (ex: ".pdf")	Yes

Child Nodes

N/A

Parent Node

<AllowedOutputTypes>

Configuring Common Reports

A <Report> node may also be defined in the /ServiceConfig/CommonReports section of the configuration and then referenced in other report groups. This allows a report to be defined once, but used multiple times. If, for instance, a particular report exists in multiple report groups, it would be beneficial to define the report in this section, and then reference it in those report groups.

For example, the following section declares a common report that can then be used anywhere else in the configuration:

```
<CommonReports configChildKey="name" >
    <Report id="ExampleCommonReport" label="Example Common Report"
isLabelTranslatable="false" reportPath="/ExampleReport/ExampleCommonReport">
        <AllowedOutputTypes configChildKey="key">
            <OutputType key="PDF" value="PDF" extension=".pdf"/>
        </AllowedOutputTypes>
```

```
    </Report>
</CommonReports>
```

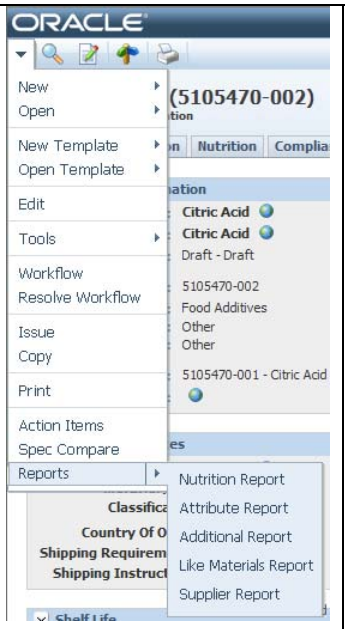
The id attribute represents the unique key that will need to be referenced if this report is to be included elsewhere. To include this report in a ReportGroup, simply declare the Report node with the idref attribute assigned to the above id. For example, the following ReportGroup declares a new report and also includes the ExampleCommonReport from the CommonReports section:

```
<Reports configChildKey="name">
    <Report name="ExampleReportA" label="Example Report A "
isLabelTranslatable="false" reportPath="/ExampleReport/ExampleReportA">
    </Report>
    <Report idref="ExampleCommonReport" />
</Reports>
```

Configuring Contextual Reporting

Individual Reports can be launched from most locations throughout the application suite (GSM, SCRM, NPD, PQS, CACS, NSM, CC, eQ Admin) by using Contextual Reporting.

Report links are added by extending the Action Navigation and adding new menu item nodes. Each new menu item entry specifies which report context, report group, and individual report should be loaded, and can send specific data to the reporting application to pre-populate specific report parameters. The visibility of each report link can be configured easily, such as specifying that the report link is only available when the user is on a Packaging specification and if the user is a member of group "Packaging Reports." See the Navigation Extensibility Guide for more information around how to add new navigational items and configure their visibility.



Clicking the report link will launch the Reporting application in a popup window, allowing the user to maintain their current context.

You can display the active object's name inside your report parameter fields. For example, if the user is on the Cajun Spice Blend specification and the report takes in a specification parameter, the string "Cajun Spice Blend" can appear in the specification parameter field as a pre-selected value. For SpecInstanceReport items, the target Report must contain a Parameter with `name="paramObjectPKID"` using a parameter type that is one of the pop-up parameter types. Setting the parameter type this way will allow for the name of the object to be displayed. Setting the parameter type to a parameter type string will simply display the PKID itself.

For Contextual Reporting, the new navigation item must be assigned an XML attribute, called **ClientSideCommand**, which is configured with two possible behaviors:

1. **LaunchSpecificObjectPKIDReport** – will launch the specified report, sending the current business object's PKID and name to the Reporting Application. If the specified report has a report parameter with a name="paramObjectPKID", it will be pre-filled with the object's name for display, and the object's PKID for the value.
   Required parameters:
   a. report context name
   b. report group name
   c. report name

2. **LaunchObjectTypeReport** – will launch the specified report, sending the current business object's 4-digit object Type value to the Reporting Application. If the specified report has a report parameter with a name="paramObjectType", it will be pre-filled with the object's 4-digit type for the parameter value.
   Required parameters:
   a. report context name
   b. report group name
   c. report name

Example:

If you have a NLEA Fact Panel (report name: NLEAPanel) report inside the context Material Reports (context name: Material) and group Nutritional Reports (group name: Nutritional) and you want to call that report using contextual reporting you would use the following function:

LaunchSpecificObjectPKIDReport('Material', 'Nutritional', 'NLEAPanel');

If you want to show the navigation item on a material spec when it's in read mode, the menu item node would look something like this:

```
<MenuItem ID="GSMNavSpec" configChildKey="ID">

<MenuItem ID="NLEAPAnel" DisplayText="NLEA Panel"
ClientSideCommand="LaunchSpecificObjectPKIDReport('Material', 'Nutritional', 'NLEAPanel');"
Visible="eval:${ObjectType}==1004 &amp;amp;&amp;amp; ${IsInReadMode}" />

</MenuItem>
```

Appendix A – Creating Custom Report Parameter Types

Methods of Retrieving Data

Clients can create custom report parameter types by leveraging existing data searches found throughout the application (via EQT Models) or by calling data retriever classes (DataSources) to retrieve the necessary search data.

Displaying Search Parameters

When specifying the report parameter type in the reporting configuration, a user interface control must also be declared. The web control that will display the parameter type will depend on which of the data retrieval approaches is used.

The Reporting module provides three web controls for DataSource parameters:

ReportingControls/**LookupInputSingleSelect**.ascx – A drop-down input allowing a single selection from data specified by a DataSource
ReportingControls/**LookupInputMultiSelect**.ascx – A pop up control allowing multiple selections from data specified by a DataSource
ReportingControls/**TreeViewInput**.ascx – A pop up control for nested hierarchies, allowing a single or multiple selections from data specified by a DataSource

The EQT Model parameters all use the following web control:

ReportingControls/**EQTInput**.ascx – A pop up control with differing behavior based on which existing EQT View is used

EQT Models

The Entity Query Toolkit (EQT) is an internal framework used in Oracle Agile PLM for Process for mapping existing internal data models to search parameters and search results. This mapping is declared in configuration files that are available for reference. EQT Models define how a search occurs, including which search parameters are available, which search parameters are mandatory (and therefore hidden and always used), what data is returned, and how returned data is displayed.

Most searches throughout the application use the EQT framework. The Reporting framework can therefore leverage some existing EQT searches to create search parameters for the desired reports. However, some EQT searches used in the application may not function as needed when leveraged in EQT. For instance, one SCRM facility search allows the selection of a facility based on some search parameters. When selecting the facility, however, the facility name may not populate the web control in the reporting screen properly, even though the facility's unique ID (PKID) does get stored properly.

Caveat: The details of EQT are complex, difficult to interpret, and are comprised of many components. Additionally, EQT is an *internal* development tool only; although it may be leveraged in the Reporting

module, it is not supported as a client facing tool. Therefore, a detailed review of EQT is beyond the scope of this document.

Oracle Consulting Services may be able to provide some assistance in leveraging EQT to meet reporting needs.

Reference EQT configuration files

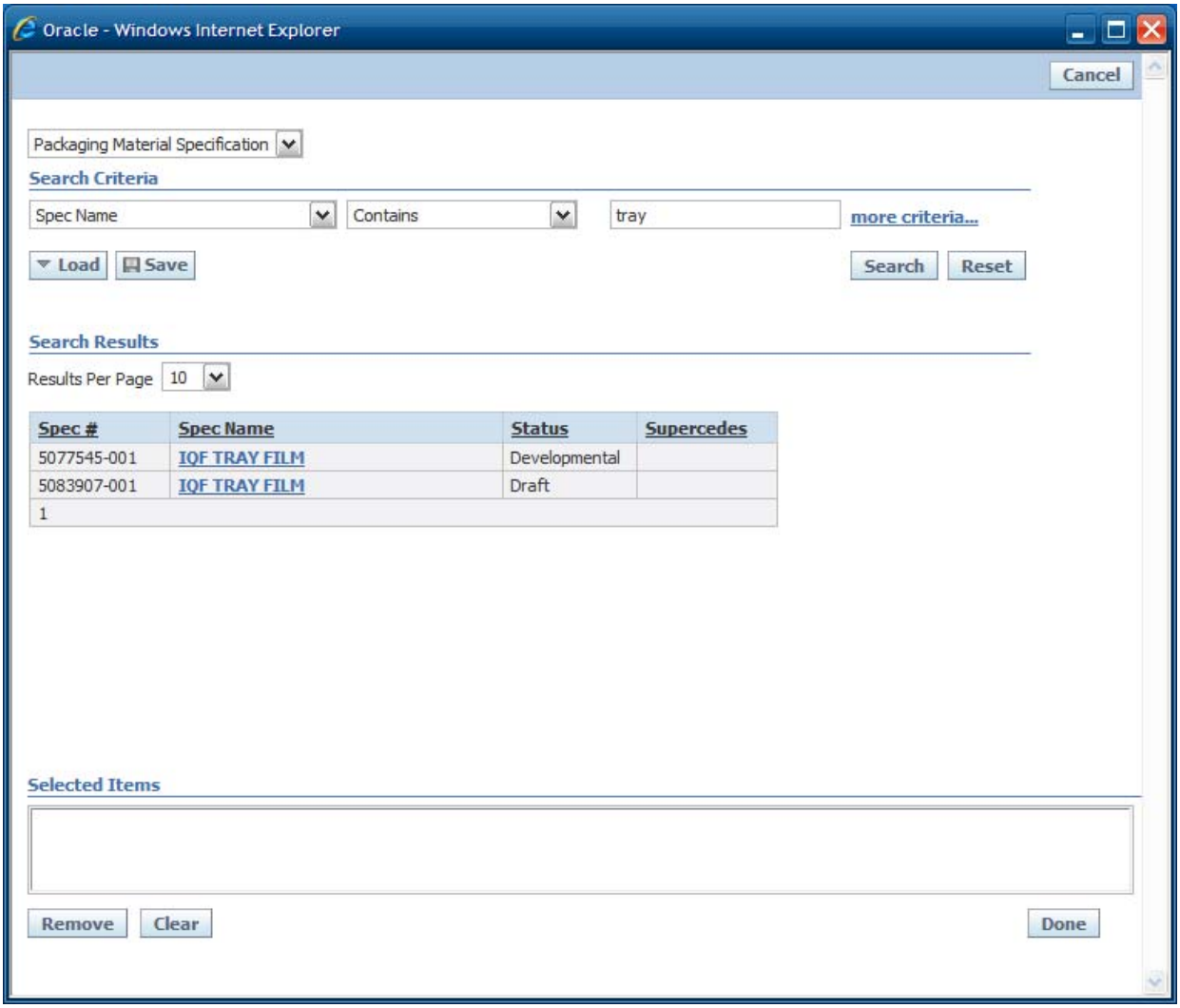
The main EQT configuration file, `EqtUIModelDefinitions.xml`, contains many of the EQT views accessible throughout the application. Some of the views may reference the views in the `EqtUICommonModels.xml` file. A reference version of each of these files is available by request.

Identifying an EQT search to use

To leverage an existing EQT search, the name of the searchable EQT view must first be identified. For searches that occur in a pop-up window, the view name may be found by looking at the search pop-up window's properties listing (right click, select **Properties**) and examining the URL. The URL may include a parameter that references the name of the EQT view.

For instance, the URL of one popup includes the following parameter:

```
&DataSource=SearchableView:Config:ProdikaSettings/SearchableMultiSelectViews,
PackagingSpecViewForProcessAndTradeSpecs
```



The EQT search view name used in this popup is `PackagingSpecViewForProcessAndTradeSpecs`.

Return value of the EQT selection

Examining the EqtUIModelDefinitions.xml file and locating this view reveals some information that will be useful:

```
<PackagingSpecViewForProcessAndTradeSpecs . . . >
  <AllowedModels>
    <Model active="true" name="Packaging Specification" alias="lblType1009"
orderByColumns="-" captionColumn="1" primaryKeyColumnName="SpecID">
```

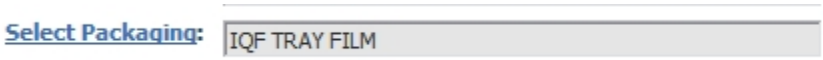
The attribute `primaryKeyColumnName` specifies which field is returned upon selection. In this instance, the SpecID property will be returned. This may be only partially informative, as the details of the model and its properties are hidden, but it generally can give an idea of what field value is returned.

Display value of the EQT selection

The display columns that contain an attribute `provideInSelectJS` are returned and accessible to the report parameter, so that they can fill the input box upon selection.

```
<DisplayColumns>
  . . .
  <ColumnInfo width="200" dataField="SpecName" dataFieldCaption="lblSpecName"
provideInSelectJS="true"/>
```

The columns where `provideInSelectJS` are set to true are then used by the <Model> node's `displayVariableIndex` attribute, representing the location/index of the desired property to display in the selected items input field. In this example, the SpecName field is returned as the only display item, and therefore its `displayVariableIndex` attribute would be set to 1.




```
<ParameterType type="ExampleEQTInput_PackagingSpecView"
webControl="ReportingControls/EQTInput.ascx">
   <Model
displayVariableIndex="1">SearchableView:Config:ProdikaSettings/EQTConfiguration/Search
ableMultiSelectViews,PackagingSpecViewForProcessAndTradeSpecs</Model></ParameterType>
```

Other ways to identify an EQT model

While some EQT views can be identified by the URL of the pop-up, others views are harder to identify. Examining the `EqtUIModelDefinitions.xml` file and searching for the desired information may yield results.

Oracle Consulting Services may be able to provide more assistance in identifying a desired EQT model.

Available EQT models

Currently, only the models in the following EqtUIModelDefinitions.xml sections are supported in the Reporting module:

Views in the models/ProdikaReporting node (via SearchableMultiSelectViews)
Views in the models/pqs node (via PQSSearchableMultiSelectViews)

For example, to create a report parameter to access the SpecSummaryView EQT view from ProdikaReporting, use the following entry:

```
<ParameterType type="ExampleEQTInput_SpecSummary"
webControl="ReportingControls/EQTInput.ascx">
<Model
displayVariableIndex="1">SearchableView:Config:ProdikaSettings/EQTConfiguration/Search
ableMultiSelectViews,SpecSummaryView</Model>
</ParameterType>
```

When configuring the Model node, the syntax above always begins with `SearchableView:Config:ProdikaSettings/EQTConfiguration/` and is followed by either

`SearchableMultiSelectViews` (for the models in the ProdikaReporting node) or `PQSSearchableMultiSelectViews` (for the models in the PQS node), then a comma, followed by the name of the Model.

An additional and possibly more straightforward approach to creating custom search parameters is by using DataSources.

DataSources

DataSources are classes that implement a specific interface (discussed later) whose responsibility is to retrieve search result data for selection in a pop-up web control. Clients can leverage several existing DataSources, or implement their own classes and plug them in easily. DataSources are referenced by their full class name and package name. They may also include parameters in their declarations that are then used by the class to modify its behavior. Please see *Appendix C – Object Loader URLs* and some of the examples below for more details.

Here is a report parameter that uses an existing datasource (LookupServiceListViewDataSource):

```
<ParameterType type="ExampleGenericLookup_Allergens"
webControl="ReportingControls/LookupInputMultiSelect.ascx">
   <DataSource
className="Class:Xeno.Web.UI.Controls.DataSources.LookupServiceListViewDataSource,Xeno
WebControls$AllergensTitle|Allergens"/>
</ParameterType>
```

Existing DataSources

Several existing DataSources may be leveraged to retrieve data which only require parameters to customize their results. A few are outlined here.

LookupServiceListViewDataSource

There are many data lists throughout the application, such as Languages, NPDBrands, Allergens, etc. Most lists are managed through a service that uses the database table commonLookupServiceManagers to maintain each list and how the list is configured. The ManagerID column value of this table for the desired lookup data must be passed in as the second parameter and is used by the LookupServiceListViewDataSource class to retrieve that list.

See the Oracle Agile PLM Data Training for more details on how CommonLookups work.

In the example above, the DataSource used is as follows, and takes two parameters (“AllergensTitle|Allergens”), the first for the pop-up title, and the second for the lookupManagerID.

```
<DataSource
className="Class:Xeno.Web.UI.Controls.DataSources.LookupServiceListViewDataSo
urce,XenoWebControls$AllergensTitle|Allergens"/>
```

Some lookup items are status aware, so this Datasource allows for a third parameter to be passed in to indicate if only Active items should be displayed ('ForSearching'), or all historically valid items should be displayed ('StatusAwareConsumer').

GenericListSelectView

This DataSource is used for specifying a table and column to search on, and the return value for the selected item(s). Rather than pass in the table name, however, this DataSource expects a class name which it then uses to retrieve the table name (The tablename to classname mapping can be found in the database table `orclassmetainfo`). To specify the column to retrieve, this DataSource expects the class parameter name that will then be used to retrieve the column name (The property to column name mapping can be found in the `orpropertymetainfo` table). To specify the return value column name, this DataSource expects the class parameter name that will then be used to retrieve the column name.

Example SQL to find the classname and property name:

```
SELECT tablename FROM orclassmetainfo where classname='complieswith'

SELECT columnName, propertyName FROM orpropertymetainfo
where fkORClassMetaInfo = (select pkid from orclassmetainfo where
classname='complieswith')
```

An example of using the GenericListSelectView DataSource for CompliesWith items, displays the Name of the compliesWith item, and returns the PKID for the selected item(s).

```
<ParameterType type="ExampleGenericTableLookup_CompliesWith"
webControl="ReportingControls/LookupInputMultiSelect.ascx">
   <DataSource
className="Class:Xeno.Web.UI.DataSources.GenericListSelectView,XenoWebControl
s$|CompliesWith|Name|PKID"/>
</ParameterType>
```

TaxonomyXNodeTreeViewDataSource

This DataSource is used to load multi-level taxonomies that are managed in the commonTaxonomyNamespace table. It requires a namespaceid parameter and must use the TreeViewInput control. There is also the option of making the selectable data a single- or multi-select.

An example of using this DataSource to display SCRM business units in a multi-select pop-up:

```
<ParameterType type="ExampleGenericTaxonomy_SCRMBusinessUnit"
webControl="ReportingControls/TreeViewInput.ascx">
   <DataSource
className="Class:Xeno.Web.UI.DataSources.TaxonomyXNodeTreeViewDataSource,XenoWebContro
ls$SCRMBusinessUnit" />
  <MultiSelect>true</MultiSelect>
</ParameterType>
```

Custom DataSources

Clients may also create a custom DataSource class to implement their own data retrieval operations. The class must implement the Xeno.Web.UI.Controls.IListView interface, and must include implementations of the following interfaces from the Xeno.Web.UI.Controls namespace:

The IListView interface returns an IListDataSource, which must enumerate through the results and return an IListItem for each entry.

```
public interface IListView
{
	IListDataSource ListItemsByPrefix( String prefix );
	IListDataSource ListAllItems();
	String Title{ get; }
	bool AllowAddNew{ get; }
	void AddNewListItem( IListItem item );
}

public interface IListDataSource
{
	IListItem Current{ get; }
	bool Next();
}

public interface IListItem
{
	String Value{ get; }
	String DisplayValue{ get; }
}
```

The custom DataSource class must be compiled and placed into the Web\ProdikaReporting\bin directory.

Appendix B – Applying Security to Reports

Overview

The Reporting framework allows customers to configure security rules that govern if a Report or Report Group should be visible to the current user. The `SecurityFactoryRef` attribute of the Report and ReportGroup nodes can be declared with different security plug-ins using Object Loader syntax. Each security plug-in takes parameters in its declaration that determine the access restrictions for that Report or Report Group.

Because Report nodes are children of the ReportGroup node, security is enforced hierarchically. When considering access to a Report, the user must have access to the parent Report Group.

Existing Security Plug-ins

The following report security plug-ins are available:

UserRoleBasedSecurityPluginFactory

This plug-in secures a Report, Report Group, or Report Context based on the roles to which a user belongs. The specified roles are evaluated using an OR relationship, meaning that if the user belongs to any of the roles listed, they allowed access. Negation is allowed through the use of the "!" (exclamation mark) operator as a prefix to a role name.

Example

```
SecurityFactoryRef="Class:Xeno.Prodika.SecurityModel.Contextual.UserRoleBased
SecurityPluginFactory,ProdikaLib$[SCRM_COMPANY_READER]|[SCRM_FACILITY_READER]
"
```

Note that the brackets surrounding the role name are required.

UserPropertyBasedSecurityPluginFactory

This plug-in secures a Report, Report Group, or Report Context based on the user's application access. The specified applications listed are evaluated in an OR relationship. It is important to note that this type of security considers a user's "HasAdminAccess" property, and if so, allows access regardless of the supplied permissions.

Example

```
SecurityFactoryRef="Class:Xeno.Prodika.SecurityModel.Contextual.UserPropertyB
asedSecurityPluginFactory,ProdikaLib$SCRM"
```

UserGroupBasedSecurityPluginFactory

Note that this plug-in is available in the Extensibility Pack, under the Handlers package. Be sure to add the Handlers.dll file to the web\ProdikaReporting\bin directory.

This plug-in secures a Report, Report Group, or Report Context based on the user's group access. The specified groups listed are evaluated in an OR relationship. There are several ways to specify the group: using the group name or the group PKID. Negation is allowed through the use of the "!" (exclamation mark) operator as a prefix to a group name or PKID. Additional options are detailed in the Extensibility Pack documentation.

Example

```
SecurityFactoryRef="Class:Xeno.Prodika.SecurityModel.Contextual.UserGroupBase
dSecurityPluginFactory,Handlers$Group123|Group456"
```

Appendix C – Object Loader URLs

Object Loader URLs are classpaths that are used to dynamically load objects. They are used to declare the protocol to use when loading the class, the class path, and optionally any parameters to pass to the class.

Format

[Protocol] : [Path] $ [[param1] | param2]

Protocol - Examples are "Class" and “Singleton".

Path - The fully qualified class name, including the package name. For example `"Xeno.Prodika.SecurityModel.Contextual.UserRoleBasedSecurityPluginFactory,ProdikaLib"` where ProdikaLib is the name of the package (.dll file).

When loading an object, the loader first inspects the Protocol and using lazy loading, determines an appropriate protocol handler based on this protocol’s name. The "Class" protocol may refer to a class that accepts parameters during instantiation which are defined after a "$" and delimited by "|"s (pipes).

Common Usage

The most common usage of this class is in configuration files. Often a factory class is supplied in a configuration and the Object Loader bootstraps the factory, which in turn facilitates the use of the rest of the implementation. These implementations are easily swapped by simply providing a different factory in the configuration.

Example

Class:Xeno.Prodika.Portal.WebUI.Util.Security.UserPropertyBasedSecurityPluginFactory,ProdikaLib$NPD

"Class" is the protocol, "NPD" is a parameter, and the rest of the string between the ":" and the "$" is the path as defined by the protocol. In this case, it is the class path of the object that is to be instantiated.

Appendix D – Third Party External Application Integration

The Oracle Agile Product Lifecycle Management for Process application allows customers to integrate with reports that exist in an external reporting engine. Specifically, customers can configure the application suite to gather information from within the application and provide well-formatted reports that are generated in an external system.

Currently, the PLM for Process application supports only two reporting engines:

Oracle BI Publisher

SQL Server Reporting Service

This document outlines the steps necessary to integrate the PLM for Process application with any other third party external application that can provide similar functionality.

Programmatic Interfaces

In order to achieve this integration, two classes must be created that conform to PLM for Process interfaces.

1. IServiceFactory
2. IReportGenerationService

Implementing IReportGenerationService

The PLM for Process application expects all implementations of its report integration adapters to implement the IReportGenerationService interface:

```
namespace Xeno.Prodika.Services
{
    public interface IServiceFactory
    {
        IService Create(IExecutionContext execContext, IBlockingResourcePool
servicePool);
    }
}
```

As the sample code illustrates, this class has two purposes:

i. Retrieve configuration information from the ReportingServiceConfig

ii. Initialize and return a Service that will be used to manage integration with a reporting engine (which in this case is called ParameterReportGenerationService)

```
public class ParameterReportGenerationServiceFactory : IServiceFactory
{
  private ReportingServiceConfig _config;

  public IService Create(IExecutionContext execContext, IBlockingResourcePool
servicePool)
```

```
  {
    ExtractConfiguration();
    return new ParameterReportGenerationService(_config);
  }

  private void ExtractConfiguration()
  {
    if (_config != null)
      return;
    IXMLConfigurationManager cm = (IXMLConfigurationManager)
AppPlatformHelper.ApplicationManager.EnvironmentManager.GetConfigManager("ReportingSer
viceConfig");

    _config = (ReportingServiceConfig) cm.GetConfig("ReportingServiceConfig");

    if (_config == null)
      throw new ProdikaConfigurationException("Could not read
ReportingServiceConfig");
  }
}
```

Implementing IReportGenerationService

The PLM for Process application expects all implementations of its report integration adapters to implement the IReportGenerationService interface:

```
namespace Xeno.Reporting.Service.ReportGenerationService
{
  public interface IReportGenerationService : IService
  {
    ReportingServiceConfig Config { get; }
    ReportResults GenerateReport(ReportConfig config, ITranslationCache
translationCache, string format, IDictionary parameters);
  }
}
```

This interface provides two capabilities:

i. The ability to access configuration information

ii. A method that can be converts a request for a report into the result of a report invocation.

```
internal class ParameterReportGenerationService : IReportGenerationService
{
  private readonly ReportingServiceConfig _config;
  public ParameterReportGenerationService(ReportingServiceConfig config)
  {
    _config = config;
  }

  public ReportResults GenerateReport(ReportConfig config, ITranslationCache
translationCache, string format, IDictionary parameters)
  {
    ReportResults result = new ReportResults();
    result.ReportContent = BuildParameterReport(parameters);
    result.MimeType = format;
    return result;
  }
```

```
  private static byte[] BuildParameterReport(IDictionary parameters)
  {
    StringBuilder builder = new StringBuilder();
    foreach (DictionaryEntry parameter in parameters)
    {
      builder.AppendFormat("Parameter '{0}' has value: '{1}'<br/>", parameter.Key,
parameter.Value);
    }
    return new ASCIIEncoding().GetBytes(builder.ToString());
  }

  public ReportingServiceConfig Config
  {
    get { return _config; }
  }
  public IServiceContext ServiceContext
  {
    get { return new ServiceContextAdapter(); }
    set { ; }
  }
}
```

In our example, the configuration information was passed into this class upon creation. All that needs to be done is to provide it back in order to fulfill the contract of the IReportGenerationService.

The generation is also reasonably straightforward. The following information is provided to the method that implements this behavior:

i. The ReportConfig, which contains information on which report is being requested as well as its service location

ii. The translation cache that has been configured for this particular report

iii. The format of the result

iv. All parameters that have been picked by a user and are being passed into the report

As can be seen, the sample code in question does not make a call to a reporting engine. Instead, it displays all parameters that have been provided to it in HTML.

A more realistic implementation would access the Name and ServiceLocation values off the ReportConfig object and then use that to drive which report needed to be invoked.

It should be noted that the approach taken by PLM for Process does not assume a specific protocol (SOAP/DB/etc). That is a decision best left up to the implementer.

Configuration Changes

In order for the PLM for Process application to use custom code for report integration, we must register it at the appropriate location. In this particular case, the correct location is the Config/Custom/CustomerSettings.config file.

For our sample implementation, the entry for SQLReportingService in the configuration file would be modified to this if the following assumptions about the class that implements IServiceFactory were true:

i. It is named ParameterReportGenerationServiceFactory and is in namespace ParameterReportingServiceAdapter

ii. It is contained in the ParameterReportingServiceAdapter.dll

iii. The DLL file is available to all web applications (i.e., it is present in each Web/*/bin directory)

```
<SQLReportingService refscope="Session"
factory="Class:ParameterReportingServiceAdapter.ParameterReportGenerationServiceFactor
y,ParameterReportingServiceAdapter"
  configChildKey="name"
  configAttributeOverrideModifier="Replace"  > </SQLReportingService>
```

PLM for Process must be configured with the user name and password in order integrate properly. To set this value, please refer to the “Setup Assistant” section of the *Agile Product Lifecycle Management for Process Configuration Guide*.