Systinet

Software Version: 10.03
Windows and Linux Operating System

Report Editor User Guide

Document Release Date: April 2017
Software Release Date: April 2017

Hewlett Packard
Enterprise

Legal Notices

Warranty

The only warranties for Hewlett Packard Enterprise products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Hewlett Packard Enterprise shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from Hewlett Packard Enterprise required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

© 2003-2017 Hewlett Packard Enterprise Development LP

Trademark Notices

Adobe™ is a trademark of Adobe Systems Incorporated.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

UNIX® is a registered trademark of The Open Group.

This product includes an interface of the 'zlib' general purpose compression library, which is Copyright © 1995-2002 Jean-loup Gailly and Mark Adler.

Documentation Updates

To check for recent updates or to verify that you are using the most recent edition of a document, go to: https://softwaresupport.hpe.com/.

This site requires that you register for an HP Passport and to sign in. To register for an HP Passport ID, click **Register** on the HPE Software Support site or click **Create an Account** on the HP Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HPE sales representative for details.

Support

Visit the HPE Software Support site at: https://softwaresupport.hpe.com/.

Most of the support areas require that you register as an HP Passport user and to sign in. Many also require a support contract. To register for an HP Passport ID, click **Register** on the HPE Support site or click **Create an Account** on the HP Passport login page.

To find more information about access levels, go to: https://softwaresupport.hpe.com/web/softwaresupport/access-levels.

HPE Software Solutions Now accesses the HPSW Solution and Integration Portal website. This site enables you to explore HPE Product Solutions to meet your business needs, includes a full list of Integrations between HPE Products, as well as a listing of ITIL Processes. The URL for this website is https://softwaresupport.hpe.com/km/KM01702731.

About this PDF Version of Online Help

This document is a PDF version of the online help. This PDF file is provided so you can easily print multiple topics from the help information or read the online help in PDF format. Because this content was originally created to be viewed as online help in a web browser, some topics may not be formatted properly. Some interactive topics may not be present in this PDF version. Those topics can be successfully printed from within the online help.

Contents

Chapter 1: Report Editor Guide

Welcome to the *Report Editor User Guide*. This guide explains how to use Report Editor as part of HPE Systinet.

This guide contains the following chapters:

Chapter 2: Report Editor

Workbench includes Report Editor enabling you to create and deploy new reports easily.

> **Tip:** HPE Systinet Report Editor is an implementation of the Built-In Reporting Tool (BIRT), an open source Eclipse-based reporting system. The Help menu includes the BIRT documentation. The *BIRT Report Developer Guide* describes the general use of BIRT to create reports. The Report Editor Guide adds to this by explaining how to use this functionality in conjunction with HPE Systinet.

This chapter introduces Report Editor in the following sections:

- "Overview " below
- "User Interface " on the next page
- "Use Cases " on page 11

Overview

Report Editor interacts with HPE Systinet, enabling you to access existing report definitions, create customized reports, and deploy them to the HPE Systinet server.

To access, create, and deploy reports with HPE Systinet, follow these steps:

1. **Create a reporting project.**

   For details, see "Configuring a Reporting Project " on page 13.

2. **Create and modify reports.**

   For details, see "Designing Reports " on page 18.

3. **Deploy your reports to HPE Systinet.**

   For details, see "Deploying Reports " on page 53.

4. **Access your reports in HPE Systinet.**

   For details, see:

   - "Use Cases " on page 11

- ○ "Accessing Reports " on page 53

- ○ "Example: Failure Impact Report for the Reports Tab " on page 57

- ○ *HPE Systinet User Guide*

User Interface

The default platform perspective is split into a number of views as following:

- **Project Explorer**

  The tree view of your reporting project. For details, see "Project Explorer " on the next page.

- **Data Explorer**

  The view of the data sources and data sets for a particular report. For details, see "Data Explorer " below.

- **Server Explorer**

  A list of HPE Systinet servers connected to HPE Systinet Workbench. For details, see "Server Explorer " on page 10.

- **Editor Views**

  The main area of the perspective contains report editor views. In this view you can edit the layout and properties of your report definition.

Data Explorer

The Data Explorer displays the sources and sets of data used by particular report you are working with.

Data Explorer contains the following folder structure:

- **Data Sources**

  Open the context menu and select **New Data Source** to access a database.

  For details, see "Adding a Data Source to a Report " on page 18.

- **Data Sets**

  Open the context menu and select **New Data Set** to define the data used by the report.

For details, see "Creating a Data Set " on page 19.

Project Explorer

The Project Explorer is a view of the projects in your workspace and the report definitions they contain. This view is also an interaction point with HPE Systinet.

Each project contains a set of report definitions and the HPE Systinet library.

Each report definition contains the following sections:

- **Overview**

  Double-click to open a properties editor view enabling you to change the name and description, and to mark the report as **Shared** to all users or leave the report as user specific.

- **Report Design**

  Double-click to open a layout editor view enabling you to change the appearance and content of the report definition.

The Project Explorer contains additional context menu options enabling you to interact with a running HPE Systinet server. Right-click the project name or a particular report definition, and select **HPE Systinet** to view the options listed in "Project Context Menu Options" and "Report Definition Context Menu Options".

Project Context Menu Options

Option	Function
Download Reports	Import report definitions from HPE Systinet. For details, see "Importing Report Definitions " on page 16.
Upload to Server	Export a report to the default HPE Systinet server. For details, see "Deploying a Report to HPE Systinet" on page 54.
Update from Server	Update report definitions from HPE Systinet.
Remove from Server	Delete reports from HPE Systinet.
Upload To Other Server	Export a report to a specified HPE Systinet server.
Build Extension	Create a reporting extension for HPE Systinet containing all the reports in your project. For details, see "Creating a Reporting Extension " on page 54.

Report Definition Context Menu Options

Option	Function
Upload to Server	Export a report to the default HPE Systinet server. For details, see "Deploying a Report to HPE Systinet" on page 54.
Update from Server	Update report definitions from HPE Systinet. For details, see "Importing Report Definitions " on page 16.
Remove from Server	Delete the report from HPE Systinet.
Upload To Other Server	Export a report to a specified HPE Systinet server.
Build Extension	Create a reporting extension for HPE Systinet containing all the reports in your project. For details, see "Creating a Reporting Extension " on page 54.

Server Explorer

The Server Explorer displays the HPE Systinet servers connected to HPE Systinet Workbench. The functionality is shared by all the HPE Systinet Workbench editors.

Right-click a server in the Server Explorer to open the context menu described in "Server Explorer Context Menu Options".

Server Explorer Context Menu Options

Option	Function
New Server	Add a server for downloading assertions and taxonomies (Assertion Editor, Taxonomy Editor, and Customization Editor).
Remove Server	Delete a server from the Server Explorer.
Download Taxonomy	Download a taxonomy from a platform server (Taxonomy Editor and Customization Editor).
Download Assertion	Download assertions from a platform server (Assertion Editor).
Download Report	Download reports from a reporting server (Report Editor).
Properties	View and edit the server name, URL, username, and password.

Use Cases

HPE Systinet enables you to add custom BIRT reports created in Report Editor to the Reports tab.

To create and use a custom BIRT report in HPE Systinet:

1. Create a new report definition.

   For details, see "Creating a Report Definition " on page 18.

2. Add a data source for the report.

   For details, see "Adding a Data Source to a Report " on page 18.

3. Create a data set for the report,

   For details, see "Creating a Data Set " on page 19.

4. Deploy the report to HPE Systinet.

   For details, see "Deploying a Report to HPE Systinet" on page 54.

5. Add the report to the HPE Systinet Reports Tab.

   For details, see the "Custom Birt Reports" section of the *HPE Systinet User Guide*.

For a step-by-step walkthrough demonstrating this use case, see "Example: Failure Impact Report for the Reports Tab " on page 57.

Chapter 3: Getting Started

HPE Systinet is distributed with a set of predefined reports and various ways to access them.

Report Editor provides a mechanism to customize these existing reports, and also enables you to create new report definitions from scratch.

This chapter describes the setup of Report Editor in the following sections:

- "Installing a Report Development Environment " below

- "Configuring a Reporting Project " on the next page

Installing a Report Development Environment

This section describes a setup to develop and test reports on a single computer.

For supported platforms, see `readme.txt` in the installation folder.

A typical report development environment consists of the following:

- PC or notebook with at least 2GB RAM and at least 5GB free disk space

- Windows XP or Windows 2000

- Oracle eXpress Edition (XE)

  For deployment details, see "Oracle XE Account Setup " on the next page.

  You do not need to install XE if you have dba-level access to deployed HPE Systinet environment using Oracle 10g.

- Java 1.5_9 or higher

  For deployment details, see "Java Environment Setup " on the next page.

- HPE Systinet deployed to JBoss

  You do not need to install HPE Systinet if you have access to a test environment.

- Oracle Developer

HPE Software recommends this tool for tuning SQL queries.

Java Environment Setup

After installing Java, you must set an environment variable, JAVA_HOME. HPE Systinet Workbench requires Java SE Development Kit (JDK) 1.8.0 (64 bit version only) or higher. You must include the path to this version of the JDK in the **JAVA_HOME** environment variable.

To set the **JAVA_HOME** environment variable:

1. In Windows, click **Control Panel > System and Security > System > Advanced system settings > Environment Variables**.

   The Environment Variables dialog box is displayed.

2. In the User Variables section, click **New**.

3. Input Variable Name as **JAVA_HOME** and set Variable Value to the installation folder of Java SE Development Kit (JDK) 1.8.0 or higher.

4. Click **OK** to exit each dialog box.

Oracle XE Account Setup

If you install Oracle XE, you must create a user account.

To create an Oracle XE account:

1. Open the XE web console and log on as the system user.

2. Create a new user, **platform**, with the password **changeit**, then grant the new user all privileges.

Configuring a Reporting Project

To use Report Editor with HPE Systinet, you must create or import a project, and import any report definitions you want to modify.

These procedures are described in the following sections:

- "Creating a Reporting Project " on the next page

- "Importing a Reporting Project " on the next page

- "Importing Report Definitions " on page 16

In addition you can configure Report Editor in the following ways:

- "Rebranding HPE Systinet Reports " on page 16

- "Configuring the Model " on page 17

- "Configuring the Reporting Logging Level " on page 17

Creating a Reporting Project

The Report Editor organizes your work into project folders in your workspace.

To create a reporting project:

1. Do one of the following:

   a. In the HPE Systinet Workbench Welcome page, select **Create Report Project**.

   b. From the menu, select **File > New > Report Project**.

   c. Press **Alt+Shift+N**, and then press **R**. The New Project window is displayed.

      Expand **HPE Systinet**, select **Report Project**, and then click **Next**.

   The New Report Project Wizard is displayed.

2. In the New Report Project dialog box, enter the following parameters:

Parameter	Definition
Project Name	The name of your report project.
Create from Existing Extension	Select this option if you want to create a new project from a previous report extension. If selected, input the path or browse for the location of the report extension.
Use Default Location	If selected, Report Editor stores the project in your default workspace. If not selected, input the path or browse for an alternative workspace.

3. Do one of the following:

   a. Select **Create a New Server**, and then click **Next**. Go to Step 4.

   b. Select **Use an Existing Server**, select the server from the list and input its credentials, and then click **Next**. Go to Step 5.

4. In the New Server dialog box, add the required parameters, and then click **Next**.

   For parameters descriptions, see "New Report Project Wizard: New Server " on page 62.

5. If needed, in the New Data Source dialog box, click **Manage Drivers** to add a JDBC driver.

   For details, see "Adding a JDBC Driver " on page 47.

6. In the New Data Source dialog box, add the required parameters.

   For parameters descriptions, see "New Report Project Wizard: Default Library Configuration - Create a new data source " on page 62.

   If your database server is running, click **Test Connection** to check if your settings are correct.

7. Click **Next** to set project options.

8. Do one of the following:

   a. To create a generalized reporting project, click **Finish** and exit the wizard.

   b. To create a project associated with a specific report category, click **Next** and continue this procedure.

      The HPE Systinet server must be running to access report definitions.

9. Select a report category for your reporting project, and then click **Next**.

10. Select report definitions from the selected category to download to the Report Editor, and then click **Finish**.

Importing a Reporting Project

You can import a reporting project created elsewhere into your HPE Systinet Workbench.

To import a reporting project:

1. Copy the project folder to your local files system.

2. From the menu, select **File > Import**.

   The Import dialog box is displayed.

3. Expand **General**, select **Existing Projects into Workspace**, and then click **Next**.

4. Click **Browse**, select the project from your local file system, and then click **OK**.

5. Click **Finish** to import the reporting project.

Importing Report Definitions

The HPE Systinet server contains all the report definitions for HPE Systinet. To modify existing reports, you must import their definitions from the HPE Systinet server.

To import HPE Systinet report definitions:

1. In the Project Explorer, open the context menu for the project, and do one of the following:

   a. Select **HPE Systinet > Download Reports** to import from the current HPE Systinet server.

   b. Select **Import**. Expand **HPE Systinet** and select **Report**, and then click **Next**.

      Select a reporting server. The HPE Systinet server must be running. Click **Next**.

2. Select the report category and click **Next**.

3. Select the report definitions you require and click **Finish**.

Rebranding HPE Systinet Reports

By default, most HPE Systinet reports include an HPE branded logo. You can replace this image with your own company-specific image.

To rebrand HPE Systinet reports:

1. Open `s2_default.rptlibrary` for editing.

2. In the Outline view, expand Embedded Images.

3. Open the context menu for `logoHP.gif` and select **Delete**.

4. Open the context menu for Embedded Images and select **New Embedded Image**.

5. Upload your new company branded image which must be named `logoHP.gif`.

6. Save `s2_default.rptlibrary`.

7. Redeploy all reports that use this image to HPE Systinet. For details, see "Deploying Reports " on page 53.

Configuring the Model

HPE Systinet divides the architecture model into a public and system model. By default, Report Editor accesses the public model which means that you can only create reports that access these public artifact types.

If you need to access system artifact types in your reports then you must modify the library.

To Extend the Reporting Model to access System Artifacts:

1. Open the `s2_default.rptlibrary` file in your workspace in a text editor.

2. Locate the `oda-data-source` element with `name="dqlDatabase"` in the file.

3. Edit the `property name="odaURL"` element, adding `|model=sys` to the property value.

4. Save `s2_default.rptlibrary`.

Configuring the Reporting Logging Level

HPE Systinet reporting uses the BIRT engine for report execution which uses standard JDK logging with root logger `org.eclipse.birt`. You can configure the logging in the same way as any other component that uses JDK logging, using a configuration file passed with the **-Djava.util.logging.config=** switch.

The default logging level is set to SEVERE, but this is overidden if you specify the logging level in a logging configuration file.

For details about JDK logging, see http://java.sun.com/j2se/1.5.0/docs/guide/logging/overview.html.

Chapter 4: Designing Reports

This chapter describes the functionality of Report Editor in conjunction with HPE Systinet.

This chapter contains the following sections:

- "Creating a Report Definition " below

- "Adding a Data Source to a Report " below

- "Creating a Data Set " on the next page

- "Designing a Report Layout " on page 20

- "Shared and User-Specific Reports " on page 21

- "Modifying Existing Report Definitions " on page 22

Creating a Report Definition

The first step in defining a report is to create the report definition.

To create a report definition:

1. Select **File > New > Report** from the menu.

2. Input a name and description for the report and click **Next**.

3. Select the project folder and click **Finish** to create the report definition.

4. Press **Ctrl+S** to save the report definition.

For an example of this procedure, see "Example: Failure Impact Report Layout " on page 58.

Adding a Data Source to a Report

By default, new reports contain DQL and SQL data sources setup during project creation. If you need to use a different one, you must add it to the report.

HPE Systinet supports only the default data sources. Reports using user-created data sources work within Report Editor but do not display when the report executes on the HPE Systinet server.

To add a data source to a report:

1. Open the report definition editor view.

2. In the Data Explorer, open the Data Sources context menu and select **New Data Source**.

   The New Data Source dialog box is displayed.

3. Do one of the following:

   a. To add a datasource for DQL queries, select **DQL JDBC Data Source** and click **Next**.

   b. To add a datasource for SQL queries, select **JDBC Data Source** and click **Next**.

4. Optionally, if not already installed, to add a JDBC driver, you can click Manage Drivers in the Data Source dialog box.

   > **Note:** The DQL driver is pre-installed. Ensure that you do not add driver more than what is required.

   For details, see "Adding a JDBC Driver " on page 47.

5. In the Data Source dialog box, add the required parameters.

   For parameter descriptions, see "New Report Project Wizard: Default Library Configuration - Create a new data source " on page 62.

   If your database server is running, click **Test Connection** to check if the settings are correct.

6. Click **Finish** to add the data source to your report.

7. Press **Ctrl+S** to save your changes to the report definition.

Creating a Data Set

Each report is associated with one or more queries and each query returns a data set.

To create a data set:

1. From the Report Design perspective, open the report definition view for the report that requires new data set.

   If you need to use an alternative data source, see "Adding a Data Source to a Report " on the previous page.

2. In the Data Explorer, open the Data Sets context menu, and select **New Data Set**.

   The New Data Set dialog is displayed.

3. In the New Data Set dialog box, add the required parameters.

    For parameters descriptions, see "New Data Set " on page 61.

4. Click **Next** to open the **Query** dialog box.

5. Do one of the following:

    a. Copy a query from your development tool to the query editor. Go to Step 8.

    b. Create your query in Report Editor as described in Step 6 to Step 7.

6. Use **Schema** (SQL only), **Filter**, and **Type**, and then click **Apply Filter** to focus on the data you require in Available Items.

    For SQL queries, HPE Systinet Workbench limits the amount of data collected. To change the settings, see "Changing the Data Collection Settings " on page 47.

7. Browse Available Items to find the data items you require. Drag and drop items from Available Items to the query editor and construct your query.

    For portability to different HPE Systinet installations using different schemas, remove the schema name from the variables in your query if your database schema does not require them.

8. Click **Finish** to add the data set to the report definition.

9. Press **Ctrl+S** to save your changes.

For details about writing DQL queries, see "Using DQL" on page 23.

For details about the database structure and writing SQL queries, see "Exploring the Database " on page 46.

For an example of this procedure, see "Example: Failure Impact Data Set " on page 57.

> **Note:** If possible, use only one dataset per report definition. If you need more datasets in one report (for example, multiple properties in one table row), use JavaScript to build one table.

Designing a Report Layout

You can create layout for your report using the **Layout** tab of editor view.

To design a report layout:

1. In the blank area of your *<your_report>*.rptdesign editor view, right-click and select **Insert > Label**.

   where, *<your_report>* is the name of your report given while creating the report definition. For example, failure_impact.rptdesign.

2. Enter a heading for your report. Click the **Property Editor** tab, to format the heading.

3. In the blank area of your reports editor view, right-click and select **Insert > Table**.

4. Input the table parameters and select the data set.

5. Enter the column heading for the table.

6. In the Data Explorer, expand the required Data Set for the report. Drag the data set field from the Data Explorer to the table cell in the report editor view.

   Repeat the above step to link all the columns of the table in report editor view with the data set.

7. Optionally, as per the requirement, you can insert dynamic text, data, image, grid, list or chart in the blank area of your report editor view.

8. Click the **Preview** tab to view the report you designed with data from the data source.

For more information on designing a report layout, see "Example: Failure Impact Report Layout " on page 58.

Shared and User-Specific Reports

The Report Editor enables you to determine the security of report output. A report can be marked as shared to show the same data to all users or not shared. If a report is not shared, then each user only sees the data they have read permission for in the report output.

The default value for a new report is shared.

To mark a report as shared or user-specific:

1. In the **Project Explorer**, expand the report you want to categorize and double-click **Overview** to open a view of the report definition.

2. The General Information section displays a Shared check-box.

   Select **Shared** to show the same report output to all users. De-select **Shared** to make the report output restricted according to the user read permissions.

3. Press **Ctrl+S** to save your changes.

Modifying Existing Report Definitions

Report definitions that already exist in HPE Systinet can be modified.

To modify existing report definitions:

1. Import the report definition.

    For details, see "Importing Report Definitions " on page 16.

2. To open the report definition, in the Project Explorer, double-click the report definition *rptdesign*.

3. Make your modifications, as described in "Creating a Report Definition " on page 18.

4. Press **Ctrl+S** to save your changes.

5. Redeploy the report definition to HPE Systinet.

    For details, see "Deploying a Report to HPE Systinet" on page 54.

Chapter 5: Using DQL

The DQL query language provides a simple query solution for the System Data Model (SDM). It enables you to query all aspects of the model – artifacts, properties, relationships, governance, and compliance.

This chapter describes DQL in the following sections:

- "Introduction to DQL" below

- "DQL Reference" on page 32

- "DQL with 3rd Party Products" on page 42

Introduction to DQL

DQL is an SQL-like language that enables you to query the repository of artifacts in HPE Systinet defined by the SDM model. DQL preserves SQL grammar, but uses artifacts instead of tables, and artifact properties instead of table columns. As DQL is based on SQL you can apply your SQL knowledge to DQL.

A simple example is to return the name and description of all business service artifacts.

```
select name, description
  from businessServiceArtifact
```

In HPE Systinet, you can use DQL queries in the following use cases:

- To create reports in HPE Systinet Report Editor.

- To customize pages of the HPE Systinet user interface.

- You can also use DQL in any SQL designer using the DQL JDBC driver. For more details, see "DQL in SQL Designers" on page 44

The following sections contain DQL examples:

- "Artifact Inheritance" on the next page

- "Categorization Properties" on the next page

- "Complex Properties" on page 26

Artifact Inheritance

Artifacts in HPE Systinet form a hierarchy defined by the SDM model. Artifacts lower in the hierarchy inherit properties from higher abstract artifact types. `artifactBase` is the root abstract artifact type in the SDM hierarchy. All other artifacts are below it in the hierarchy and inherit its properties. You can query abstract artifacts and return a result set from all the instances of artifact types lower in the hierarchy.

Property groups function in a similar way, querying a property group returns results from all artifact types that inherit properties from the group.

The following query returns results from all implementation artifacts; SOAP Services, XML Services, and Web Applications.

```
select name, serviceName
  from implementationArtifact
```

Notice that in this query, serviceName is a specific property of SOAP Service artifacts. In the result set, name is returned for all implementation artifacts but serviceName is only returned for SOAP service artifacts. For other implementation types, the serviceName is NULL.

> **Caution:** Different artifact types may define the same properties with different cardinalities. In cases where two artifact types define the same property with different cardinality, querying a shared parent abstract artifact for these properties may fail. Examples that fail include **SELECT environment FROM artifactBase** and **SELECT accessPoint FROM artifactBase**.

Categorization Properties

Categorization properties are a special case of complex properties.

Categorization properties have the following sub-properties:

- `val` - machine readable name of the category.

- `name` - human readable name of the category.

- `taxonomyURI` - identifies the taxonomy defining the category set.

  > **Note:** taxonomyURI is not defined for named category properties.

HPE Systinet uses categorization properties in the following ways:

- **Named category properties** (for example, business service criticality).

  The following query returns the names, descriptions, and versions of all business service artifacts which are categorized using the named criticality categorization property with a high failure impact.

  ```
  select name, description, version
    from businessServiceArtifact
    where criticality.val =
          'uddi:systinet.com:soa:model:taxonomies:impactLevel:high'
  ```

  > **Note:** taxonomyURI is not defined for named category properties. The name of the category property implies the taxonomy.

- **categoryBag**

  `categoryBag` is a complex property that includes sub-property `categories` which is a categorization property and `categoryGroups`. `categoryGroups` also contains categorization sub-property `categories` and a `taxonomyURI` defining the meaning of the group. HPE recommends querying `_category` instead of `categoryBag` to ensure that all categories are queried.

  The following query returns the names, descriptions, and versions of all business service artifacts which are categorized by the Gift certificate category (14111608) of the `uddi:uddi.org:ubr:categorization:unspsc` taxonomy.

  ```
  select name, description, version
    from businessServiceArtifact
    where categoryBag.categories.taxonomyURI =
          'uddi:uddi.org:ubr:categorization:unspsc'
      and categoryBag.categories.val = '14111608'
  ```

- **identifierBag**

  `identifierBag` is a complex property similar to `categoryBag` that includes sub-property `categories`. `identifierBag` does not contain the `categoryGroups` subproperty. HPE recommends querying `_category` instead of `identifierBag` to ensure that all categories are queried.

- **_category**

This generic categorization property holds all categorizations from `categoryBag`, `identifierBag`, and all named categorization properties from the given artifact type.

The following query returns the names, descriptions, and versions of all business service artifacts which are categorized with a high failure impact.

```
select name, description, version
 from businessServiceArtifact
 where _category.val =
       'uddi:systinet.com:soa:model:taxonomies:impactLevel:high'
   and _category.taxonomyURI =
       'uddi:systinet.com:soa:model:taxonomies:impactLevel'
```

> **Caution:** When you use the generic `_category` property you must specify the taxonomy using the `_category.taxonomyURI` sub-property. When you use a named categorization property the taxonomy is implicitly known and does not need to be specified.

Complex Properties

Complex properties are composed of one or more single or multiple-valued sub-properties (for example, address contains sub-properties addressLines in multiple cardinality, country in single cardinality, and so on. The sub-property addressLines is also a complex sub-property, containing a value and useType.). It is only possible to query the sub-property components of primitive types. Components of sub-properties are separated by dot (**.**) and in MS Access you can use **$** as a separator.

```
select address.addressLines.value, address.country
 from personArtifact
 where address.city = 'Prague'
```

Embedding SQL Queries

DQL works with SDM entities (artifacts and properties) only and cannot directly access database tables. In some cases, it is necessary to obtain values from outside the SDM (for example, system configuration). You can use an SQL subquery in a NATIVE clause of a DQL query. By default, DQL expects SQL to return an unnamed single column of values.

The following example returns business services owned by the administrator using the name defined during installation:

```
select name,description, version
 from businessServiceArtifact
```

```
where _owner in (
  native {select svalue from systemConfiguration
            where name='shared.administrator.username'})
```

You can use NATIVE clauses instead of expressions, as a condition in WHERE clauses, as a column in SELECT clauses, and as a artifact reference in FROM clauses. For details, see "DQL Grammar" on page 36 .

If you use a NATIVE clause to formulate part of a FROM clause, you must specify parameters to bind columns defined by SQL to properties used by DQL.

Each parameter consists of the following:

- The property name defines how DQL addresses columns returned from the NATIVE SQL statement.

- The property type which may be returned by the metadata of a column is optional and if not specified is assumed to be a text string.

The parameters are enclosed in brackets in the native clause, delimited by commas, and the type is separated from the name using whitespace.

The following example shows a query with NATIVE SQL in a DQL FROM clause.

```
select B.p_id, B.s_val, A.name, B.state_index
  from (
    native(s_val, s_name, state_index integer, p_name, p_id)
      {select S.val as s_val, S.name as s_name, S.state_index as state_index,
            P.name as p_name, P.id as p_id
        from rylf_state S, rylf_process P
        where S.fk_rylf_process=P.id and P.name='Application Lifecycle'}) B
  left join artifactBase A on A._currentStage.val = B.s_val
  order by B.p_id, B.state_index
```

The NATIVE statement returns the following columns; s_val, s_name, p_name, and p_id of type String, and state_index of type Integer.

Note: Native clauses can not contain variables (? or :<variable>).

Fixing Multiple Properties

Consider a business service with keywords, 'Finance' and 'Euro'. The intuitive query for finding a 'Euro Finance' service is as follows:

```
select name, description, version
```

```
   from businessServiceArtifact b
 where b.keyword.val = 'Finance'
    and b.keyword.val = 'Euro'
```

This query does not work as a single instance of keyword can never be both 'Finance' and 'Euro'

The solution is to fix instances of multiple properties as shown in the following query:

```
select name, description, version
  from businessServiceArtifact b, b.keyword k1, b.keyword k2
 where k1.val = 'Finance'
    and k2.val = 'Euro'
```

Modifiers

Modifiers define primary sets of objects (artifacts and their revisions) to query. If no modifier is specified, the last revisions of undeleted artifacts for which the user has read access are queried.

The following modifiers are available:

- Revision related modifiers (mutually exclusive):

    ○ **all_rev** - queries all revisions of artifacts.

    ○ **last_approved_revision** - queries the last approved revisions of artifacts.

- Security related modifiers (mutually exclusive):

    ○ **my** - queries artifacts belong to the user.

    ○ **writable** - queries artifact the user has write permission for.

    ○ **no_acl** - queries all artifacts regardless of security.

- Other modifiers:

    ○ **include_deleted** - queries all instances, including deleted artifacts.

You can use multiple comma-separated modifiers.

The following query returns all business services that you own that are marked as deleted.

```
select b.name, b.version, b.keyword.name
  from businessServiceArtifact b (my, include_deleted)
 where _deleted = '1'
```

Primitive Properties

Primitive properties are simple properties, such as numbers, characters, and dates, that may occur once or multiple times for an artifact depending on the cardinality as defined in the SDM. For example, in the SDM Model, each person is represented by a person artifact. The person artifact includes a name property with single cardinality and an email property with multiple cardinality.

The following query returns the name and all emails for each person in the repository.

```
select name, email
  from personArtifact
```

Instances of primitive properties with multiple cardinality are all returned as comma separated values. For example, all the emails for a person return as a concatenated, comma-separated string. If there is no instance of the property for an artifact, a null value is returned.

The following query returns the name, description, and version of all business service artifacts whose version is 2.0.

```
select name, description, version from businessServiceArtifact where version = '2.0'
```

> **Note:** By default, DQL queries return the latest revisions of artifacts unless you specify revision modifiers. For details, see "Modifiers" on the previous page.

Relationships

A relationship is a special kind of complex property pointing to another artifact. HPE Systinet uses relationships to join artifacts.

The following queries are semantically identical and return all business services and the contact details of their provider. These queries do not return business services that do not have providers.

- The following query is an example of an *SQL92-like* join which uses the USING clause.

```
select b.name, b.version, b.keyword.name, p.name as contact, p.email
  from businessServiceArtifact b
  join personArtifact p using provides
```

The relationship property `provides` leads from person artifacts to business service artifacts is specified after the `using` keyword.

- The following query is an example of an *SQL92-like* join which uses the ON clause.

```
select b.name, b.version, b.keyword.name, p.name as contact, p.email
  from businessServiceArtifact b
  join personArtifact p on bind(provides)
```

The relationship property `provides` leads from person artifacts to business service artifacts is specified with the `bind` predicate in the `WHERE` clause.

- The following query is an example of an *old-style* join which uses the BIND predicate.

```
select b.name, b.version, p.name as contact, p.email
  from businessServiceArtifact b, personArtifact p
  where bind(p.provides, b)
```

The BIND predicate specifies that the `provides` relationship of the person artifact points to business service artifacts.

The following query also returns all business services and the contact details of their provider. This query is an example of a LEFT JOIN. The LEFT JOIN extends the previous queries by returning business services that do not have providers.

```
select b.name, b.version, p.name as contact, p.email
  from businessServiceArtifact b
  left join personArtifact p using provides
```

Each relationship has the following sub-properties which you can query:

- rType - the SDM QNames of the relationship type.

- useType - the values of the useType relationship property.

- target - the UUIDs of the artifact the relationship points to (deprecated).

It is possible to specify a particular provider type using useType. The following queries return all business services and their contact details where the provider is an architect.

```
select b.name, b.version, p.name as contact, p.email
  from businessServiceArtifact b, personArtifact p
  where bind (p.provides, b)
    and p.provides.useType = 'architect'
```

```
select b.name, b.version, p.name as contact, p.email
  from businessServiceArtifact b
  join personArtifact p on bind(p.provides, b)
    and p.provides.useType = 'architect'
```

It is possible to traverse several relationships using several old-style joins or SQL-92-like join clauses in the same query. The following example queries business services in applications, which are also part of a project.

```
select b.name, b.description, a.name as Application, p.name as Project
```

```
  from businessServiceArtifact b
  join hpsoaApplicationArtifact a using hpsoaProvidesBusinessService
  join hpsoaProjectArtifact p using contentRelationshipType
```

In cases where artifacts may be joined by multiple properties, you can use a generic `_relation` property together with the additional `rType` condition.

```
select A.name as A_name, B.name as B_name
  from hpsoaApplicationArtifact A left join artifactBase B on bind(A._relation)
    and A._relation.rType in (
        '{http://systinet.com/2005/05/soa/model/property}
hpsoaProvidesBusinessService',
        '{http://systinet.com/2005/05/soa/model/property}r_providesBusinessProcess'
    );
```

You can use the `target` relationship sub-property to bind the source and target of a relationship.

```
select b.name, b.version, p.name as contact, p.email
  from businessServiceArtifact b, personArtifact p
    where p.provides.target = b._uuid
```

> **Caution:** The `target` property and this style of comparison is deprecated and its use is not recommended. Use the `bind` predicate instead.

Virtual Properties

DQL defines virtual properties that are not defined by the SDM. HPE Systinet stores or calculates these properties enabling DQL to query meta information about artifacts. These virtual properties provide information about lifecycle, compliance, domains, etc.

The following example returns lifecycle details from the last approved revisions of all business service artifacts, ordered by lifecycle stage.

```
select name, _lastApprovedStage.name Stage, _revision
  from businessServiceArtifact(last_approved_revision)
  order by Stage
```

The following example returns the name and compliance status of last approved revisions of all business services which has a compliance status of at least 80%.

```
select b.name, b._complianceStatus
  from businessServiceArtifact b (last_approved_revision)
  where b._complianceStatus >= 80
```

HPE Systinet repository content exists within a domain structure where each artifact exists within only one domain. The default functionality of DQL queries all domains but HPE Systinet provides virtual

properties enabling you to query artifacts within a particular domain. The following example returns business service names and the domain details of all business service artifacts that exist within the EMEA domain.

```
select A.name, A._domainId, A._domainName
  from businessServiceArtifact A
  where A._domainId="EMEA"
```

DQL provides the following macros for querying within domain hierarchies:

- **#SUBDOMAINS('*domainId*')**

  Queries the specified domain and all its sub-domains.

- **#SUPERDOMAINS('*domainId*')**

  Queries the specified domain and all its parent domains.

The following query returns all business services in the EMEA domain and any of all of its sub-domains.

```
select A.name
  from businessServiceArtifact A
  where A._domainId in #SUBDOMAINS('EMEA')
```

The following query returns the name and virtual properties artifactTypeName and owner from the latest revisions of consumer properties (the property group for all consuming artifact types).

```
select name, _artifactTypeName, _owner
  from consumerProperties
```

For details of all virtual properties, see "Properties in DQL" on the next page.

DQL Reference

This section provides a reference to properties and DQL grammar in the following sections:

- "Properties in DQL" on the next page

- "DQL and SQL" on page 36

- "DQL Grammar" on page 36

Properties in DQL

Artifact (property group) properties hold values which may be queried in DQL expressions.

DQL recognises the following properties:

- SDM Properties

  Properties defined in the SDM Model. For details, see *System Data Model* section in the *HPE Systinet Customization Guide.*

- Virtual, System, and Other Properties

  Properties holding metadata about artifact instances.

Properties may be one of the following:

Property Kind	Description
Primitive	Holds string, number, or boolean values. For example, name, description, version. A primitive property is defined in DQL statements by the artifact type name or alias, the property delimiter (. or $), followed by the property name. For example, personArtifact.name. The artifact name or alias is optional (with the delimiter) when the property is specific to a single artifact type in the query.
Complex	Hold complex structures such as address. Only primitive sub-properties of complex properties may be queried. Properties and sub-properties are separated by . or $. For example, `personArtifact.address.city`.
Categorization	Hold categorization data and are handled in a similar way to complex properties. Categories consist of `name`, `val`, and `taxonomyURI` components. For example, `businessServiceArtifact.criticality.name`.
Relationships	Properties that specify a directional relationship to other artifacts.

All values that you can query are of a particular data type. The following table describes these data types, and gives examples of how to use them in a query.

Data Type	Description	Example
Number	Numeric values	_revision = 1
String	Text values	_revisionCreator = 'admin'
Date Time	Date and Time (ms since 00:00 1/1/1970)	_revisionTimestamp > 1274447040124 /* revisions made since 15:04 21/5/2010 */
Boolean	True or False flags	_deleted = '1'

Properties may have the following cardinalities:

Property Cardinality	Description
Single	Only one instance of the property exists for an artifact and it may be optional or required.
Multiple	The property is a list of values and so occurs multiple times for an artifact. In WHERE clauses, using multiple properties returns particular artifacts if any instance of the multiple property matches the condition. In SELECT clauses, using multiple properties returns all instances of the multiple property as a concatenated, comma-separated string.

DQL uses the following system properties:

System Property	Description
_artifactTypeName	The human readable name of the artifact type (the SDM label of the artifact). HPE recommends using _sdmName in conditions, and _artifactTypeName in SELECT clauses.
_category	All categorizations for an artifact with `name`, `val`, and `taxonomyURI` components.
_deleted	The deletion marker flag (boolean).
_id	The database ID of an artifact instance (number, deprecated - use _uuid).
_longDescription	HPE Systinet supports a long description including HTML tags up to 25000 characters by default. HPE recommends using the `description` property in DQL queries instead as queries using _longDescription may affect performance and the HTML tags may corrupt report outputs. `description` contains only the first 1024 text characters of `_longDescription` (may vary according to your database type). **Note:** The `platform.repository.max.description.length` property determines the maximum length of _longDescription. You can modify this property in `SYSTINET_HOME/conf/setup/configuration-propeties.xml`. DB2 has an absolute limit of 32672 characters which is sufficient for descriptions containing ASCII (single-byte encoding) characters but may be exceeded, for example by Japanese descriptions, leading to description truncation.
_owner	The user, group, or role designated as the artifact owner.
_ownerName	The human readable name of the user (taken from the use profile), group, or role artifact.
_path	Legacy REST path of the artifact (string, deprecated - use _uuid).
_relation	A generic virtual property that may be used to specify all outgoing relationships.

System Property	Description
_revision	The revision number of an artifact instance.
_revisionCreator	The user who created the revision of the artifact.
_revisionTimestamp	The date and time the revision was created.
_sdmName	The local name of the artifact type. HPE recommends using _sdmName in conditions, and _artifactTypeName in SELECT clauses.
_uuid	The unique artifact identifier.

DQL uses the following virtual properties:

Property Class	Property	Description
UI Property	_isFavorite	Marked by the user as favorite flag (boolean).
	_rating	The average rating of the artifact (double).
Security Property	_shared	Indicates that the artifact is shared and visible to users in the Sharing Principal role (boolean). For more details, see *How to Share Artifacts* in the *HPE Systinet User Guide*.
	_writable	User write permission flag (boolean). **Note:** Using this property may have a performance impact. If possible, use the **writable** modifier instead.
Contract Property	_enabledConsumer	The artifact is a valid consumer artifact type.
	_enabledProvider	The artifact is a valid provider artifact type. The artifact must also be marked as 'Ready for Consumption'.
Domain Property	_domainId	Value of the domainId property defined for the domain artifact that the artifact belongs to.
	_domainName	The readable name of the domain.

Lifecycle Property	_currentStage	Current working stage of an artifact.
	_governanceProcess	Process applicable to the artifact.
	_isApproved	Lifecycle approval flag (boolean).
	_lastApprovedRevision	Revision number of the last approved revision (number).
	_lastApprovedStage	The name of the last approved stage.
	_lastApprovalTimestamp	Timestamp for the last approval (number, ms since 00:00 1/1/1970).
	_lifecycleStatus	The status of the current lifecycle stage.
Policy Manager Property	_complianceStatus	Compliance status as a percentage (number).

DQL and SQL

DQL supports most features of SQL with the following exceptions:

- SELECT * is not supported.

- RIGHT and FULL OUTER JOIN are not supported.

- It is not possible to use properties with multiple cardinality in GROUP BY, HAVING, or ORDER BY clauses.

DQL Grammar

A DQL query consists of the following elements with their grammar explained in the following sections:

- "Select" on the next page

- "FROM Clause" on page 38

- "Conditions" on page 39

- "Expressions" on page 40

- "Lexical Rules" on page 41

Typographical Conventions

Convention	Example	Description
KEYWORDS	**SELECT**	A reserved word in DQL (case-insensitive).
parsing rules	*expr*	Name of a parsing rule. A parsing defines a fragment of DQL which consists of keywords, lexical rules, and other parsing rules.
LEXICAL RULES	*ID*	Name of a lexical rule. A lexical rule defines a fragment of DQL which consists of letters, numbers, or special characters.
[ ]	[**AS**]	Optional content.
[...]	[, *select_ item*, ...]	Iterations of optional content.
		ASC | **DESC**	Alternatives.
{ }	{ + | - }	Group of alternatives.
..	0..9	A range of allowable characters.

Select

```
select :
  subquery [ ORDER BY                    order_by_item [, order_by_item ...]]

subquery :
  subquery [ set_operatorsubquery ...]
  | (subquery)
  | native_sql
  | subquery_base

subquery_base :
  SELECT [ DISTINCT ] select_item [, select_item ...]
  FROM                  from_clause_list
  [ WHERE              condition ]
  [ GROUP BY                expression_list
    [ HAVING              condition ]
  ]

select_item :
  expr [ [ AS ] alias ]

alias :
  ID | QUOTED_ID

order_by_item :
```

```
  expr [ ASC | DESC ]

set_operator :
  UNION ALL | UNION | INTERSECT | EXCEPT

native_sql :
  NATIVE [ (column_name [ column_type ] [ , ... ] ) ]
  { sql_select }
```

Explanation:

- The { } around the sql_select are required and sql_select is an SQL query.

- The column_name and column_type specify parameters to pass from the SQL query to the DQL query.

FROM Clause

```
from_clause_list :
  { artifact_ref | subquery_ref | fixed_property | native_sql }
  [ from_clause_item ... ]

from_clause_item :
  , { artifact_ref | subquery_ref | fixed_property | native_sql }
  | [ LEFT [ OUTER ] ] JOIN
    { artifact_ref | subquery_ref } join_condition

artifact_ref :
  artifact_name [ alias ] [ (artifact_modifiers) ]

subquery_ref :
  (subquery)alias

fixed_property :
  property_refalias

artifact_modifiers :
  ID [ ,ID ... ]

artifact_name :
  ID

join_condition :
  | USINGproperty_ref
```

Conditions

```
condition :
  condition_and [ OR                          condition_and ... ]

condition_and :
  simple_condition [ AND                      simple_condition ... ]

simple_condition :
  (condition)
  | NOT                        simple_condition
  | exists_condition
  | like_condition
  | null_condition
  | in_condition
  | simple_comparison_condition
  | native_sql
  | bind

simple_comparison_condition :
  exprcomparison_opexpr

comparison_op :
 = | <> | < | > | <= | >=

like_condition :
  expr [ NOT ] LIKE                    like_expression [ ESCAPE
 STRING ]

like_expression :
  STRING
  | variable_ref

null_condition :
  expr                    IS [ NOT ] NULL

in_condition :
  expr [ NOT ] IN( { subquery | expression_list } )
  | macro

exists_condition :
  EXISTS(subquery)

bind :
  BIND(property_ref [ , alias ] )

macro :
  macro_name [ (expression_list) ]
```

```
macro_name :
  #ID
```

Explanation:

- Conditions can be evaluated to true, false, or N/A. *condition* consists of one or more *condition_and* that are connected by the **OR** logical operator.

- *condition_and* consists of one or more *simple_condition* connected by the **AND**.

- *simple_condition* is one of following:

  - *condition* in parentheses

  - Negation of *simple_condition*

  - *exists_condition*

  - *like_condition*

  - *null_condition*

  - *in_condition*

  - *simple_comparison_condition*

  - *native_sql*

- *simple_comparison_condition* is a comparison of two expressions using one of the comparison operators: =, <>, <, >, <=, >=.

- *like_condition* compares an expression with a pattern. Patterns can contain wildcards:

  - **_** means any character (including numbers and special characters).

  - **%** means zero or more characters (including numbers and special characters).

  - **ESCAPE** *STRING* is used to prefix _ and % in patterns that should represent those characters and not the wildcard.

- *alias* references the target artifact.

Expressions

```
expr :
  term [ { + | - | CONCAT } term ... ]

term :
```

```
  factor [ { * | / } factor ... ]

factor :
  (select)
  | (expr)
  | { + | - } expr
  | case_expression
  | NUMBER
  | STRING
  | NULL
  | function_call
  | variable_ref
  | property_ref
  | native_sql

case_expression :
  CASE                    case_item [ case_item ... ]
    [ ELSE                        expr ]
  END

case_item :
  WHEN                    condition                    THEN
expr

function_call :
  ID( [ DISTINCT ] { [ * ] | [ expression_list ] } )

property_ref :
  { ID | QUOTED_ID } [ { . | $ } { ID | QUOTED_ID } ... ]

expression_list :
  expr [ ,expr ... ]

variable_ref :
  ? | :ID
```

Explanation:

- Variables are of two kinds:

  - Positional variables - ? in DQL

  - Named variables - :<name_of_variable>

- When variables are used in DQL, each variable must have a value bound to the variable.

Lexical Rules

```
CONCAT :
```

```
  ||

STRING :
  [ N | n ] ' text '

NUMBER :
  [ [ INT ] . ] INT

INT :
  DIGIT [ DIGIT ... ]

DIGIT :
  0..9

ID :
  CHAR [ { CHAR | DIGIT } ... ]

CHAR :
  a..z | A..Z | _
```

Explanation:

- *ID* is sequence of characters, numbers and underscores beginning with a character or underscore.

- *QUOTED_ID* is text in quotes.

- *CONCAT* means a concatenation of strings - syntax **||**.

DQL with 3rd Party Products

DQL is provided by a JDBC driver which you can use with common SQL designers supporting 3rd-party JDBC drivers (or ODBC with an ODBC-JDBC bridge).

The following sections describe the driver and its use with third-party products:

- "DQL JDBC Driver" on the next page

- "DQL in SQL Designers" on page 44

- "DQL in MS Access" on page 44

DQL JDBC Driver

The DQL JDBC driver translates DQL queries into SQL queries and executes them using the underlying JDBC driver for the used database. The translation is provided by a remote invocation of HPE Systinet.

All the required JAR files for the DQL driver are available in `SYSTINET_HOME/client/lib/jdbc`:

- `pl-dql-jdbc.jar`

- `hessian-version.jar`

- Database driver JAR files are copied here during installation (for example, `ojdbc6.jar`).

The following table describes the driver configuration required to use the driver with 3rd party products.

DQL JDBC Driver Configuration

Property	Description
Connection String	jdbc:systinet:http(s)://<username>@<host:port>/<context>[|schema=schema name][|model=list of allowed models] • `<username>` is the HPE Systinet username who executes the DQL query using HPE Systinet permissions security. • `<host:port>` are the connection details of your HPE Systinet installation (for example, localhost:8080 for HTTP or secure:8443 for HTTPS). • `<context>` is the application server context, the default is `em`. • `|schema=schema name` is the schema of the user who owns HPE Systinet database tables. This parameter is optional. When omitted, it is supposed that the user account used to access the database is also the owner of HPE Systinet tables. In case a common user or read-only user is used, use the power user schema name, unless the DQL JDBC Driver cannot provide metadata regarding artifacts and properties. • `|model=list` of allowed models. This parameter is optional and represents a comma-separated list of models. Only artifacts from allowed models are provided in JDBC metadata as tables. The available models are `sys` and `public`. By default, artifacts from the public model are provided. For example, `jdbc:systinet:http://admin@demoserver.acme.com:8080/soa|schema=SOA320`
DB Credentials	The database username and credentials used for direct access to the HPE Systinet database. In most cases it is the user who owns all tables for HPE Systinet - called the *power user*. In case of "Manual Database Arrangement" with a power user and a common user (who has only read/write access to tables, but can not create other tables), use the common user account. In case the common user is still too powerful

Property	Description
	to be shared, the DB administrator can create another - "read-only user" with read-only access to HPE Systinet tables. Note that the read-only user must also have created synonyms/aliases for HPE Systinet tables to pretend that HPE Systinet tables are in the schema of the read-only user. For more details, see *Database Installation Types* in the *HPE Systinet Installation and Configuration Guide*.
DQL JDBC Classname	com.hp.em.dql.jdbc.DqlDriver

Note: The DQL JDBC driver must be able to connect to the database from the client. Use the full hostname for your database used during installation or setup. In the event of connection problems, verify the firewall settings between the local server and the database server.

DQL in SQL Designers

SQL Designer software can use the DQL driver if the designer is JDBC-aware.

To configure a JDBC-aware SQL Designer:

1. Add the DQL JDBC JAR files to the classpath.

2. Create a JDBC connection using the properties described in "DQL JDBC Driver" on the previous page.

After you establish the DQL JDBC connection, the following functionality should be available in your SQL Designer:

- Schema introspection, browsing the list of artifact types and property groups as tables, and their properties as columns.

- DQL query execution.

DQL in MS Access

MS Access 2007 can execute DQL queries using an ODBC-JDBC bridge. Before using MS Access, you must configure the ODBC datasource in Windows.

To configure an ODBC-JDBC bridge:

1. Download and install an ODBC-JDBC bridge. For example, *Easysoft ODBC-JDBC Gateway*.

2. Configuration of:

   ○ JDBC driver configuration using the properties described in "DQL JDBC Driver" on page 43.

   ○ Bridge configuration. For details, see the documentation for the bridge software.

DQL syntax varies from the examples given in "Introduction to DQL" on page 23 in the following cases:

- Complex properties must use $ notation and be enclosed by [ ].

  `personArtifact.[address$addressLines$value], personArtifact.[address$country]`

- To use modifiers such as (include_deleted) use the Pass-Through option in MS Access.

- Left Joins do not work. Use plain Joins instead.

- For fixed properties, use the Pass-Through option in MS Access.

- For timestamps, use the Pass-Through option in MS Access.

- Native queries do not work in MS Access.

- For property aliases, do not use quoted aliases.

Chapter 6: Exploring the Database

The functionality of Report Editor relies on interaction with the database.

This chapter describes how to examine and setup the database in the following sections:

- "SDM to Database Mapping Tool " below

- "Adding a JDBC Driver " on the next page

- "Changing the Data Collection Settings " on the next page

- "Optimizing the Database " on the next page

- "Writing SQL Queries " on page 48

SDM to Database Mapping Tool

Artifacts in HPE Systinet are stored in the form of XML documents. Their structure is defined by the System Data Model (SDM). Artifacts are serialized into a database over a standard serialization layer. The serialization of data may differ from the norm, based on customer specific extensions or modifications.

The sdm2dbmap tool is a mapping tool that generates a report containing the mapping between your SDM and database tables.

To generate the report, execute the following command:

SYSTINET_HOME/lib/sdm/bin/sdm2dbmap

The mapping report is output to the following file:

`SYSTINET_HOME/lib/sdm/build/sdm2dbmap.html`

The output consists of the following parts:

- A top level 1:1 mapping between SDM artifacts and DB tables. Each artifact listed, maps directly to one table.

- A list of artifacts. Each artifact in the report maps each SDM property to a specific column in the table. There are also associated tables and foreign keys, joined using the primary key of the artifact

table.

- A report documenting the DB schema for all database tables coming from the SDM. Tables with names ending in _Rev are used to store older revisions.

Adding a JDBC Driver

To connect to a database you must specify a JDBC driver.

To add a JDBC driver:

1. In the Database Connections dialog box, click **Manage Drivers**.

   The Manage JDBC Drivers dialog box is displayed.

2. Click **Add** to browse your filesystem for a driver.

   HPE Software recommends using the same JDBC driver used during HPE Systinet installation.

3. Select the driver, and then click **Open** to add it to the list of drivers in the **Manage JDBC Drivers** dialog box.

Changing the Data Collection Settings

By default, the editor limits the amount of data collected to 20 schemas and 100 tables in the Query dialog box.

To change the data collection settings:

1. From the menu, select **Windows > Preferences**.

2. Expand **Report Design > Data Set Editor**, and then select **JDBC Data Set**.

3. Change the number of schemas and tables as required, and then click **OK**.

Optimizing the Database

If you frequently use particular fields for joins in your queries, create an index for that field to improve performance.

To create an Oracle index:

1. Log on to the sqlplus or isqlplus console as the system user.

2. Enter the following command:

   create index ix_docrel_optimize2 on ry_docRel(sourceId, rtype);

   As a minimum optimization, HPE Software recommends the following:

   ○ **create index ix_res_optimize1 on ry_resource(fk_artifactType, m_deleted, id, m_path);**

   ○ **create index ix_docrel_optimize2 on ry_docRel(sourceId, rtype);**

Writing SQL Queries

Although it is possible to use the query editor in HPE Systinet Workbench, it is often more convenient to use a specialized tool to develop queries.

This section describes some typical methods and provides examples to help you design your queries.

These examples were created using Oracle Developer for use with an Oracle Database. Changes may be required to use these queries with other database types.

This section contains the following methods and examples:

- "Joining Tables in Queries " below

- "Example: Business Service SELECT " on the next page

- "Example: Contract SELECT " on the next page

- "Example: Combined Business Service and Contract SELECT " on page 50

- "Example: Accepted Contract Count for a Business Service " on page 51

Joining Tables in Queries

To join tables in your queries, add the following to the WHERE clause of your SELECT statement:

- Key or foreign key condition

- Discriminator expression

- Check the deleted flag (0 means false)

Example:

```
WHERE ryga_bsnService.id = rygt_ctgryPty.fk_resource_id
```

```
        AND rygt_ctgryPty.discriminator = 'bsnPolicy_iBag'
        AND bsnRes.m_deleted = '0'
```

Example: Business Service SELECT

This example query returns details for business services that are marked as ready for consumption:

Table ry_bsnService contains the main fields for business services.

Table ry_resource contains general information about every artifact (for all artifact types).

The inner join to ry_resource in the FROM clause excludes deleted artifacts.

```
SELECT
    bsn.id bsn_id,
        bsn.name_val bsn_name,
        bsnRes.m_path bsn_path,
        bsn.serviceVersion_val bsn_version,
        bsn.description_val bsn_description,
        bsn.productionStage_name bsn_productionStage,

FROM ryga_bsnService bsn
        INNER JOIN ry_resource bsnRes ON
        (
            bsnRes.id = bsn.id
            AND bsnRes.m_deleted = '0'
        )

WHERE bsn.readyForConsumption_val ='1'
ORDER BY lower(bsn.name_val)
```

Example: Contract SELECT

This example query returns details for contracts that have been accepted:

```
SELECT
        contract.id ctrct_id,
        contract.name_val ctrct_name,
        contract.description_val ctrct_description,
        contract.contractstate_name ctrct_state,
        providerContractRel.targetpath ctrct_targetpath
```

```
FROM ry_contract contract
      INNER JOIN ry_docrel providerContractRel ON
      (
            providerContractRel.rtype = '{http://systinet.com/
2005/05/soa/model/property}providerContractor'
            AND providerContractRel.obsolete='0'
            AND providerContractRel.deleted='0'
            AND providerContractRel.targetDeleted='0'
            AND providerContractRel.sourceId = contract.id
      )
      INNER JOIN ry_resource contractRes ON
      (
            contractRes.id =contract.id
            AND contractRes.m_deleted='0'
      )

WHERE contract.contractState_val =
'uddi:systinet.com:soa:model:taxonomies:contractAgreementStates:accepted'
```

Table ry_contract contains the main fields for contracts.

Table ry_docrel contains artifact relationship details.

The inner join to ry_docrel in the FROM clause excludes contracts where the provider-contract relationship is obsolete, deleted, or where the provider does not exist.

Example: Combined Business Service and Contract SELECT

"Example: Business Service SELECT " on the previous page and "Example: Contract SELECT " on the previous page can be combined to list the names of the contracts and the business service to which they apply.

```
SELECT
      contract.id ctrct_id,
      contract.name_val ctrct_name,
      bsn.id bsn_id,
      bsn.name_val bsn_name

FROM ryga_bsnService bsn
      INNER JOIN ry_resource bsnRes ON
      (
            bsnRes.id = bsn.id
```

```
                    AND bsnRes.m_deleted = '0'
        ),
        ry_contract contract
        INNER JOIN ry_docrel providerContractRel ON
        (
                providerContractRel.rtype = '{http://systinet.com
/2005/05/soa/model/property}providerContractor'
                AND providerContractRel.obsolete='0'
                AND providerContractRel.deleted='0'
                AND providerContractRel.targetDeleted='0'
                AND providerContractRel.sourceId = contract.id
        )
        INNER JOIN ry_resource contractRes ON
        (
                contractRes.id =contract.id
                AND contractRes.m_deleted='0'
        )
WHERE bsn.readyForConsumption_val ='1'
        AND  contract.contractState_val =
'uddi:systinet.com:soa:model:taxonomies:contractAgreementStates:accepted'
        AND providerContractRel.targetId = bsn.id
ORDER BY lower(bsn.name_val)
```

By merging the queries you create an n:1 relationship between contracts and business services.

Example: Accepted Contract Count for a Business Service

This example returns business service details and the number of accepted contracts for each service:

```
SELECT
    bsn.id bsn_id,
        bsn.name_val bsn_name,
        bsnRes.m_path bsn_path,
        bsn.serviceVersion_val bsn_version,
        bsn.description_val bsn_description,
        bsn.productionStage_name bsn_productionStage,


        ( -- Contracts --
        SELECT count (0)
        FROM ry_contract contract
```

```
            INNER JOIN ry_docrel providerContractRel ON
        (
                    providerContractRel.rtype = '{http://systinet.com

/2005/05/soa/model/property}providerContractor'
                    AND providerContractRel.obsolete='0'
                    AND providerContractRel.deleted='0'
                    AND providerContractRel.targetDeleted='0'
                    AND providerContractRel.sourceId = contract.id
        )
     INNER JOIN ry_resource contractRes ON
            (
                    contractRes.id =contract.id
                    AND contractRes.m_deleted='0'
            )
     WHERE  contract.contractState_val =
'uddi:systinet.com:soa:model:taxonomies:contractAgreementStates:accepted'
        AND providerContractRel.targetId = bsn.id
     ) ctrct_count


     FROM ryga_bsnService bsn
            INNER JOIN ry_resource bsnRes ON
            (
                    bsnRes.id = bsn.id
                    AND bsnRes.m_deleted = '0'
            )

WHERE bsn.readyForConsumption_val ='1'
ORDER BY lower(bsn.name_val)
```

Chapter 7: Deploying Reports

When your report is ready, you must deploy it to the HPE Systinet server.

Report Editor offers two methods:

- Deploy a report directly to the HPE Systinet server.

  For details, see "Deploying a Report to HPE Systinet" on the next page.

- Deploy a set of reports as an extension to the HPE Systinet server.

  This process consists of the following steps:

  a. "Creating a Reporting Extension " on the next page

  b. "Redeploying EAR File" on page 55

After you deploy the reports, see "Accessing Reports " below for detailed access reports.

You can remove custom reports from the HPE Systinet server.For details, see "Removing Report Definitions from HPE Systinet" on page 56.

Accessing Reports

"Use Cases " on page 11 describes how HPE Systinet uses specific functionality in the UI for each report category.

It is also possible to directly access the reports in the HPE Systinet server and obtain alternative outputs.

The following URLs use the Service Portfolio report as an example:

- `https://hostname:port/em/reporting/rest/reports/`

- `https://hostname:port/em/reporting/rest/reports/service_portfolio?alt=text/plain`

- `https://hostname:port/em/reporting/rest/reports/service_portfolio/documents/`

- `https://hostname:port/em/reporting/rest/reports/service_portfolio/documents/1/?alt=text/plain`

The last URL is an instance of the report. The page source contains alternate URLs. Change the URL to return a comma separated output of the report as follows:

- `https://hostname:port/em/reporting/rest/reports/service_`
  `portfolio/documents/1/content?alt=text/csv`

Creating a Reporting Extension

You can also deploy a set of reports as an extension.

To create a reporting extension:

1. In the Project Explorer, open the context menu for the project you want to deploy and select **HPE Systinet > Build Extension**.

   The Build Extension Wizard is displayed.

2. In the Build Extension Wizard, input the required parameters.

   For parameter descriptions, see "Build Extension Wizard " on page 61.

3. Click **Finish** to create the extension.

Deploy the extension to the HPE Systinet server using the Setup Tool.

Deploying a Report to HPE Systinet

The HPE Systinet plug-ins for Report Editor enable you to deploy your report directly to the HPE Systinet server.

To deploy a report to HPE Systinet:

1. Make sure that the HPE Systinet server is running.

2. In the Project Explorer or Navigator view, open the context menu for the report design you want to deploy, and select **HPE Systinet > Upload to Server**.

   You will be asked to confirm if you want to publish the default library. If you do not want this confirmation message to appear again, change the preference in the Preference menu of Report Editor.

   To change the preferences, follow this step:

   a. Expand **Window > Preferences > HPE Systinet > Report Editor** and select whether you want Assertion Editor to prompt you every time you publish, always publish the default library, or never publish the default library.

Custom Policy Manager reports do not contain **Notify** functionality by default. To add this functionality you must add the report name to the notification context file.

To add notify functionality to custom PM reports:

1. Deploy your report to HPE Systinet as described above.

2. Open `SYSTINET_HOME\deploy\hp-em.ear\lib\pm-persistence.jar\META-INF\pmNotificationContext.xml` with a text editor.

3. Add a `value` element containing the report definition name to the file.

   For example:

```
<bean id="pm-persistence.systemReportChecker"
           class="com.hp.em.policy.notification.SystemReportChecker">
   <property name="reports">
      <set>
         <!-- em system reports - report definition IDs -->
               <value>acs_basic</value>
               ...
               <value>MyTestReport</value>
         </set>
   </property>
</bean>
```

4. Save `pmNotificationContext.xml`.

5. Restart the HPE Systinet server.

Redeploying EAR File

After using the Setup Tool to apply extensions or updates, you must redeploy the EAR file to the application server. For JBoss, you can do this using the Setup Tool.

To redeploy the EAR file to JBoss:

1. Stop the application server.

2. Start the Setup Tool by executing the following command:

   SYSTINET_HOME/bin/setup.bat(sh).

3. Select the **Advanced** scenario and click **Next**.

4. Scroll down to select **Deployment** and click **Next**.

When the Setup Tool validates the existence of the JBoss Deployment folder, click **Next**.

5. Click **Finish** to close the Setup Tool.

6. Restart the application server.

Removing Report Definitions from HPE Systinet

You can remove custom report definitions from HPE Systinet.

To remove a single report, use the context menu option.

To remove custom reporting extensions:

1. Remove any custom report extensions from `SYSTINET_HOME/extensions`.

2. Execute the Setup Tool using the apply extensions scenario.

Chapter 8: Example: Failure Impact Report for the Reports Tab

To demonstrate the Reports Tab use case described in "Use Cases " on page 11, follow this example:

To create a Failure Impact Report and add it to the a Reports Tab:

1. Create the failure impact report definition.

    Follow the procedure described in "Creating a Report Definition " on page 18 with the following inputs:

Parameter	Definition
Name	Failure Impact
Description	Summary of Business Service Failure Impact Status

2. Create the Failure Impact data set.

    For details, see "Example: Failure Impact Data Set " below.

3. Create the Failure Impact Report layout.

    For details, see "Example: Failure Impact Report Layout " on the next page.

4. Deploy the Failure Impact Report to HPE Systinet.

    For details, see "Deploying a Report to HPE Systinet" on page 54.

5. Add the Failure Impact Report to the HPE Systinet Reports Tab.

    For details, see *Custom Birt Reports* section in *HPE Systinet HPE Systinet User Guide*.

6. Select **Failure Impact** from the list.

Example: Failure Impact Data Set

To demonstrate the reporting tool use case described in "Creating a Data Set " on page 19, follow this example:

To create the Failure Impact data set:

1. Follow the procedure described in "Creating a Data Set " on page 19 and input the following parameters in the New Data Set dialog box:

Parameter	DQL Option	SQL Option
Data Set Name	Failure Impact	
Data Source	dqlDatabase	Database
Data Set Type	DQL Select Query	SQL Select Query

2. Use the filter to display only the data that you want to use.

   For SQL queries, all standard tables in HPE Systinet begin with RYGA.

3. Use Available Items to explore the data and identify the tables and columns you need for your query.

   For this report all the information required is for business services:

   ○ For DQL, artifact type **businessServiceArtifact**.

   ○ For SQL, table **RYGA_BSNSERVICE**.

4. Construct your query by typing in the query input area and dragging and dropping items from **Available Items**.

   For this report, the DQL query is:

   ```
   select bs.criticality.name as "Failure Impact Status",          count
   (bs.criticality.name) as "No. of Services"  from businessServiceArtifact bs
   group by bs.criticality.name
   ```

   For this report the SQL query is:

   ```
   select RYGA_BSNSERVICE.CRITICALITY_NAME as "Failure Impact Status",
   count(RYGA_BSNSERVICE.CRITICALITY_NAME) as "No of Services"  from RYGA_
   BSNSERVICE  group by RYGA_BSNSERVICE.CRITICALITY_NAME
   ```

5. Click **Finish** to complete your query and view the Output Columns for the data set.

6. Click **Preview Results** to view the actual data in your database.

7. Click **OK** and press **Ctrl+S** to save the data set as part of the report.

Example: Failure Impact Report Layout

To demonstrate the reporting tool use case referred to in "Designing a Report Layout " on page 20, follow this example:

To Create the Layout of the Failure Impact Report:

1. In the failure_impact.rptdesign editor view, right-click and select **Insert > Label**.

2. Type the report heading `Business Service Failure Impact Status`.

3. Use the Property Editor view to change the format of the heading.

4. In a blank area of the failure_impact.rptdesign editor view, right-click and select **Insert > Table**.

5. Input the table parameters and select the data set. In this case, 2 columns and the **Failure Impact** data set.

6. Select the left column header row, right-click and select **Insert > Text**. Input column header `Failure Impact Status`, and then click **OK**. Repeat for the right column header, `No of Services`.

Business Service Failure Impact Status	
Failure Impact Status	**No of Services**
Detail Row	
Footer Row	

⊞ Table

7. In the Data Explorer, expand **Data Sets > Failure Impact**.

8. Drag **Failure Impact Status** from the Data Explorer to the left detail cell. Repeat for **No of Services** and the right detail cell.

Business Service Failure Impact Status	
Failure Impact Status	**No of Services**
[Failure Impact S...]	[No of Services]
Footer Row	

9. In a blank area of the failure_impact.rptdesign view, right-click and select **Insert > Chart** to open the New Chart dialog box.

10. As this report requires a pie chart, select **Pie Chart** and click **Next**.

11. Under Select Data Set, select **Use Data Set** and **Failure Impact** from the drop-down list.

12. Drag **Failure Impact Status** from **Data Preview** to the **Category Definition** input.

13. Drag **No of Services** from **Data Preview** to the **Slice Size Definition** input.

14. Click **Next** to format the chart appearance.

15. Select **Chart Area**, amend **Chart Title** to `Business Service Failure Impact Status`, and then click **Finish** to add the chart to the report definition.

16. Drag the chart area outline to the size you require, and then press **Ctrl+S** to save the report definition.

17. Select the **Preview** tab to view the report with data from your data source.

Appendix A: Dialog Boxes

Each Report Editor input dialog is described in the following sections:

- "Build Extension Wizard " below. Creating a report extension.

- "New Data Set " below. Creating a new data set.

- "New Report Project Wizard " on the next page. Creating a new report project.

Build Extension Wizard

Enter general parameters for the report extension.

Option	Definition
Name	The name of the reporting extension.
Description	A description for the reporting extension.
Include library in extension	Select this check-box if you want to include the default library in the extension.
Folder	The location of the reporting extensions folder.
Filename	The filename of the reporting extension.

New Data Set

Enter data set name, data source, and data set type for the new data set.

Parameter	Definition
Data Set Name	Input a name for the data set.
Data Source	Select the data source from the drop-down list. The selection of a DQL or SQL datasource determines the exact query language required for your query.
Data Set Type	Choose one of the options from the drop-down list: - Select Query

Parameter	Definition
	• Stored Procedure Query

New Report Project Wizard

The New Report Project Wizard consists of the following stages depending on the options you select:

1. "New Report Project Wizard: Default Library Configuration - Create a new data source " below

2. "New Report Project Wizard: New Server " below

New Report Project Wizard: Default Library Configuration - Create a new data source

Configure the new data source and specify your login credentials.

Parameter	Definition
Server URL	The location of the HPE Systinet server.
Server Username	Your HPE Systinet log in credentials.
Driver Class	Automatically filled with the corresponding data base name on the server.
Database Username and Password	Your database credentials.
Database Schema	The database schema name.
Manage Drivers	Add, delete, or restore JAR files, and edit drivers.
Test Connection	Test the server connection using the specified parameters.

New Report Project Wizard: New Server

Create a new server for the project and optionally specify your authentication credentials.

Parameter	Definition
Name	The name of the server to display in the report editor.
URL	The location of the HPE Systinet server.
Username and Password	Credentials for access to the HPE Systinet server.
Save credentials	Select to store the input credentials.
Validate connection	Select to validate the server connection. The HPE Systinet server must be running to validate.

Appendix B: Troubleshooting

This appendix describes some common problems and their resolutions:

- "Incomplete Table List " below

- "The Views Missing from Platform Perspective " below

- "The Report Result does not Appear " on the next page

- "Oracle Developer Queries Cause Errors in HPE Systinet Workbench " on the next page

- "SQL Error on Report Previews " on page 66

- "HPE Systinet Workbench Looks Different from the Pictures in the Documentation " on page 66

- "HPE Systinet Workbench Displays Scrambled Table Names " on page 66

Incomplete Table List

Problem

The list of tables is incomplete. HPE Systinet Workbench limits the number of schemas and tables accessed for performance reasons.

Solution

Change the settings, as described in "Changing the Data Collection Settings " on page 47.

The Views Missing from Platform Perspective

Problem

There are views missing from the Platform perspective. The most likely reason for this is that some of the views are closed.

Solution

Reset the platform perspective as follows:

1. From the menu, select **Window > Reset Perspective**.

The Report Result does not Appear

Problem

The report result is not deployed to the HPE Systinet server.

Solution

Verify the report result as follows:

1. In your browser, navigate to the latest report instance page. For example,
    `https://hostname:port/reporting/reports/service_`
    `portfolio/documents/1/?alt=text/plain`.

2. Check the following status elements:

    ◦ `<category label="finished"`
      `scheme="urn:com:systinet:reporting:execution:status" term="finished"/>`

    ◦ ` <category label="report document" scheme="urn:com:systinet:reporting:kind"`
      `term="urn:com:systinet:reporting:kind:document"/>`

    A correctly finished report contains the *finished* value.

If the report execution does not finish, check the JMS settings, JMS queue, or the reporting service logs to analyze the bottleneck.

Oracle Developer Queries Cause Errors in HPE Systinet Workbench

Problem

HPE Systinet Workbench returns errors for imported queries. The most likely reason is that your table names do not include the schema.

Solution

Include the schema name in the **FROM** clause of your query. For example, `soadb.ry_docrel`.

SQL Error on Report Previews

Problem

When viewing the Preview page of a report, the following SQL error is displayed:

```
ReportDesign (id = 1):
+ Cannot get the result set metadata.
SQL statement does not return a ResultSet object.
SQL error #1: Invalid column name ROWNUM
```

This error occurs because the Preview is not aware of the database type that the query is applicable to.

Solution

Check the database type as follows:

1. In the Preview page, click **Show Report Parameters**.

2. Verify that the database-type parameter matches the database you use.

HPE Systinet Workbench Looks Different from the Pictures in the Documentation

Problem

HPE Systinet Workbench does not look exactly as depicted in the user documentation. The most likely reason for this disparity is that you are in the wrong perspective.

Solution

Switch to the platform perspective as follows:

1. From the menu, select **Window > Open Perspective > Platform**.

HPE Systinet Workbench Displays Scrambled Table Names

Problem

The tables names are not readable. These are most likely to be deleted tables stored in Oracle 10gR2.

Solution

Either change the maximum number of tables to 3000, as described in "Changing the Data Collection Settings " on page 47, or purge the deleted tables.

To purge deleted tables:

1. Log on to the sqlplus ot isqlplus console as the system user.

2. Enter the following commands:

    ○ **PURGE RECYCLEBIN**

    ○ **PURGE DBA_RECYCLEBIN**