

HP Business Service Management

For the Windows, Linux operating systems

Software Version: 9.21

RTSM Developer Reference Guide

Document Release Date: November 2012

Software Release Date: November 2012



Legal Notices

Warranty

The only warranties for HP 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. HP 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 HP 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

© Copyright 2005 - 2012 Hewlett-Packard Development Company, L.P.

Trademark Notices

Adobe™ is a trademark of Adobe Systems Incorporated.

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

AMD and the AMD Arrow symbol are trademarks of Advanced Micro Devices, Inc.

Google™ and Google Maps™ are trademarks of Google Inc.

Intel®, Itanium®, Pentium®, and Intel® Xeon® are trademarks of Intel Corporation in the U.S. and other countries.

Java is a registered trademark of Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates.

UNIX® is a registered trademark of The Open Group.

Acknowledgements

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

This product includes software developed by Apache Software Foundation (<http://www.apache.org/licenses>).

This product includes OpenLDAP code from OpenLDAP Foundation (<http://www.openldap.org/foundation/>).

This product includes GNU code from Free Software Foundation, Inc. (<http://www.fsf.org/>).

This product includes JiBX code from Dennis M. Sosnoski.

This product includes the XPP3 XMLPull parser included in the distribution and used throughout JiBX, from Extreme! Lab, Indiana University.

This product includes the Office Look and Feels License from Robert Futrell (<http://sourceforge.net/projects/officelnfs>).

Documentation Updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates or to verify that you are using the most recent edition of a document, go to:

<http://h20230.www2.hp.com/selfsolve/manuals>

This site requires that you register for an HP Passport and sign in. To register for an HP Passport ID, go to:

<http://h20229.www2.hp.com/passport-registration.html>

Or click the **New users - please register** link 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 HP sales representative for details.

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.

This document was last updated: Friday, November 16, 2012

Support

Visit the HP Software Support Online web site at:

<http://www.hp.com/go/hpsoftwaresupport>

This web site provides contact information and details about the products, services, and support that HP Software offers.

HP Software online support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support web site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract. To register for an HP Passport ID, go to:

<http://h20229.www2.hp.com/passport-registration.html>

To find more information about access levels, go to:

http://h20230.www2.hp.com/new_access_levels.jsp

Contents

RTSM Developer Reference Guide	1
Contents	6
Creating Discovery and Integration Adapters	14
Adapter Development and Writing	15
Adapter Development and Writing Overview	16
Content Creation	17
The Adapter Development Cycle	17
Startup and Preparation of Copy	18
Development and Testing	19
Cleanup and Document	19
Create Package	19
Data Flow Management and Integration	19
Associating Business Value with Discovery Development	20
Researching Integration Requirements	21
Developing Integration Content	24
Developing Discovery Content	26
Discovery Adapters and Related Components	26
Separating Adapters	27
Implement a Discovery Adapter	28
Step 1: Create an Adapter	31
Step 2: Assign a Job to the Adapter	38
Step 3: Create Jython Code	40
Configure Remote Process Execution	41
Discovery Content Migration Guidelines	42
Discovery Content Migration Guidelines Overview	43
Version 9.x New Infrastructure Features	44
Guidelines for Developing Cross-Data Model Scripts	47
Implementation Tips	48

Package Migration Utility	49
Troubleshooting and Limitations	50
Developing Jython Adapters	51
HP Data Flow Management API Reference	52
Create Jython Code	53
Use External Java JAR Files within Jython	53
Execution of the Code	54
Modifying Out-of-the-Box Scripts	54
Structure of the Jython File	54
Imports	55
Main Function – DiscoveryMain	55
Functions Definition	56
Results Generation by the Jython Script	57
The ObjectStateHolder Syntax	57
The Framework Instance	59
Finding the Correct Credentials (for Connection Adapters)	62
Handling Exceptions from Java	63
Support Localization in Jython Adapters	64
Add Support for a New Language	64
Change the Default Language	65
Determine the Character Set for Encoding	66
Define a New Job to Operate With Localized Data	66
Decode Commands Without a Keyword	67
Work with Resource Bundles	68
API Reference	69
Fields	69
Arguments	70
Work with Discovery Analyzer	71
Tasks and Records	71
Logs	71
Run Discovery Analyzer from Eclipse	77
Record DFM Code	87

Jython Libraries and Utilities	89
Error Messages	92
Error Messages Overview	93
Error-Writing Conventions	94
Error Severity Levels	97
Developing Generic Database Adapters	98
Generic Database Adapter Overview	99
TQL Queries for the Generic Database Adapter	100
Reconciliation	102
Hibernate as JPA Provider	103
Prepare for Adapter Creation	106
Prepare the Adapter Package	111
Upgrade the Generic DB Adapter from 9.00 or 9.01 to 9.02 and Later	114
Configure the Adapter – Minimal Method	115
Configure the Adapter – Advanced Method	118
Implement a Plugin	123
Deploy the Adapter	126
Edit the Adapter	127
Create an Integration Point	128
Create a View	129
Calculate the Results	130
View the Results	131
View Reports	132
Enable Log Files	133
Use Eclipse to Map Between CIT Attributes and Database Tables	134
Adapter Configuration Files	140
The adapter.conf File	141
The simplifiedConfiguration.xml File	142
The orm.xml File	143
The reconciliation_types.txt file	152
The reconciliation_rules.txt File (for backwards compatibility)	152
The transformations.txt File	154

The discriminator.properties File	155
The replication_config.txt File	156
The fixed_values.txt File	156
The persistence.xml File	156
Out-of-the-Box Converters	158
Plugins	162
Configuration Examples	163
Simplified Definition	163
Advanced Definition	164
Simplified Definition	166
Advanced Definition	168
Simplified Definition	168
Advanced Definition	169
Simplified Definition	170
Advanced Definition	171
Adapter Log Files	172
External References	174
Troubleshooting and Limitations	175
Developing Java Adapters	176
Federation Framework Overview	177
SourceDataAdapter Flow	180
SourceChangesDataAdapter Flow	180
PopulateDataAdapter Flow	180
PopulateChangesDataAdapter Flow	181
Adapter and Mapping Interaction with the Federation Framework	182
Federation Framework for Federated TQL Queries	183
Interactions between the Federation Framework, Server, Adapter, and Mapping Engine	185
Federation Framework Flow for Population	194
Adapter Interfaces	196
OneNode Interfaces	196
Data Adapter Interfaces	196

Pattern Topology Interfaces (Deprecated as of UCMDB 9.00)	197
Additional Interfaces	197
Adapter Interfaces for Synchronization	197
Debug Adapter Resources	198
Add an Adapter for a New External Data Source	199
Implement the Mapping Engine	206
Create a Sample Adapter	208
XML Configuration Tags and Properties	208
Developing Push Adapters	211
Developing Push Adapters Overview	212
Differential Synchronization	213
Prepare the Mapping Files	214
Write Jython Scripts	217
Support Differential Synchronization	221
Build an Adapter Package	223
Mapping File Schema	225
Mapping Results Schema	237
Viewing KPIs in External Applications	240
Set Up an Adapter to View KPIs in an External Application	241
Using APIs	243
Introduction to APIs	244
APIs Overview	245
HP Universal CMDB API	246
Conventions	247
Using the HP Universal CMDB API	248
General Structure of an Application	249
Put the API Jar File in the Classpath	251
Create an Integration User	252
HP Universal CMDB API Reference	254
Use Cases	255
Examples	257
RTSM (HP Universal CMDB) Web Service API	258

Conventions	259
RTSM (HP Universal CMDB) Web Service API Overview	260
RTSM (HP Universal CMDB) Web Service API Reference	262
Call the Web Service	263
Query the RTSM	264
Update the RTSM	267
Query the BSM Class Model	269
getClassAncestors	269
getAllClassesHierarchy	269
getCmdbClassDefinition	270
Query for Impact Analysis	271
UCMDB General Parameters	272
UCMDB Output Parameters	275
UCMDB Query Methods	277
executeTopologyQueryByNameWithParameters	277
executeTopologyQueryWithParameters	278
getChangedCIs	279
getCINeighbours	279
getCIsById	280
getCIsByType	280
getFilteredCIsByType	281
getQueryNameOfView	284
getTopologyQueryExistingResultByName	284
getTopologyQueryResultCountByName	285
pullTopologyMapChunks	285
releaseChunks	287
UCMDB Update Methods	288
addCIsAndRelations	288
addCustomer	289
deleteCIsAndRelations	289
removeCustomer	289
updateCIsAndRelations	289

UCMDB Impact Analysis Methods	291
calculateImpact	291
getImpactPath	291
getImpactRulesByNamePrefix	292
Use Cases	293
Examples	295
The Example Base Class	295
Query Example	296
Update Example	308
Class Model Example	312
Impact Analysis Example	314
Adding Credentials Example	316
Internal-Global ID Conversion API	320
Using the Internal-Global ID Conversion API	321
How to Convert RTSM Internal IDs to Global IDs	322
Parameters	322
Example	322
How to Convert Global IDs to RTSM Internal IDs	325
Parameters	325
Example	325
Data Flow Management API	327
Data Flow Management API Overview	328
Conventions	329
Data Flow Management Web Service	330
Call the Web Service	331
Data Flow Management Methods	332
Data Structures	332
Managing Discovery Job Methods	333
Managing Trigger Methods	334
Domain and Probe Data Methods	336
Credentials Data Methods	338
Data Refresh Methods	340

Code Sample	342
-------------------	-----

Part 1

Creating Discovery and Integration Adapters

Chapter 1

Adapter Development and Writing

This chapter includes:

Adapter Development and Writing Overview	16
Content Creation	17
Developing Integration Content	24
Developing Discovery Content	26
Implement a Discovery Adapter	28
Step 1: Create an Adapter	31
Step 2: Assign a Job to the Adapter	38
Step 3: Create Jython Code	40
Configure Remote Process Execution	41

Adapter Development and Writing Overview

Prior to beginning actual planning for development of new adapters, it is important to understand the processes and interactions commonly associated with this development.

The following sections can help to enable you to successfully manage and execute a discovery development project.

This chapter:

- Assumes a working knowledge of Run-time Service Model and some basic familiarity with the elements of the system. It is meant to assist you in the learning process and does not provide a complete guide.
- Covers the stages of planning, research, and implementation of new discovery content for Run-time Service Model, together with guidelines and considerations that need to be taken into account.
- Provides information on the key APIs of the Data Flow Management Framework. For full documentation on the available APIs, see the *HP UCMDB API Reference*. (Other non-formal APIs exist but even though they are used on out-of-the-box adapters, they may be subject to change.)

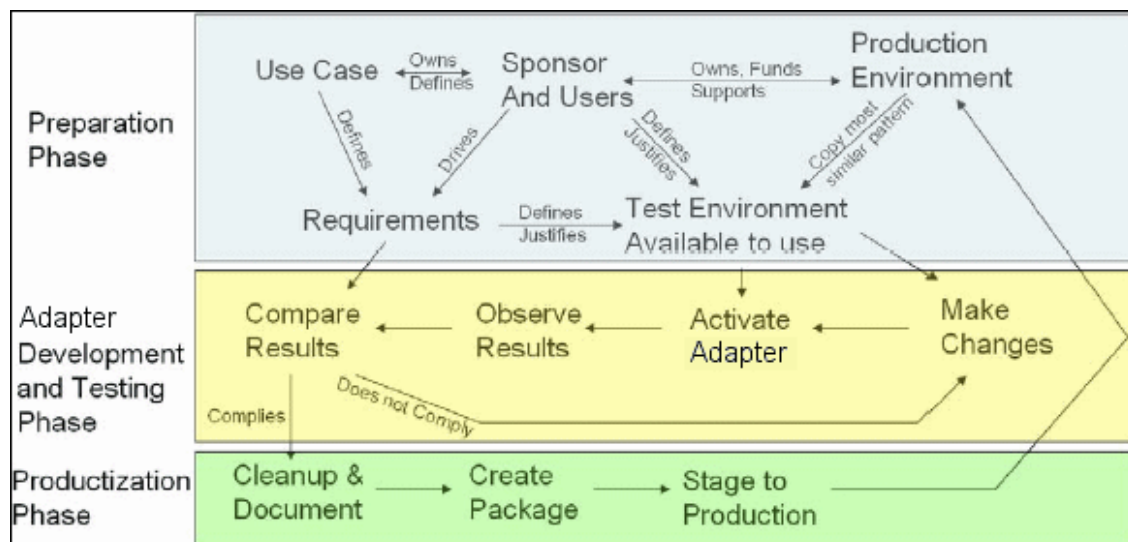
Content Creation

This section includes:

- "The Adapter Development Cycle" below
- "Data Flow Management and Integration" on page 19
- "Associating Business Value with Discovery Development" on page 20
- "Researching Integration Requirements" on page 21

The Adapter Development Cycle

The following illustration shows a flowchart for adapter writing. Most of the time is spent in the middle section, which is the iterative loop of development and testing.



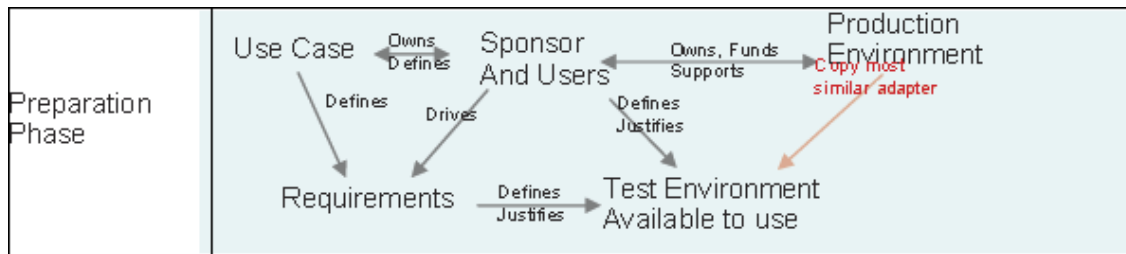
Each phase of adapter development builds on the last one.

Once you are satisfied with the way the adapter looks and works, you are ready to package it. Using either the BSM Package Manager or manual exporting of the components, create a package *.zip file. As a best practice, you should deploy and test this package on another BSM system before releasing it to production, to ensure that all the components are accounted for and successfully packaged. For details on packaging, see "Package Manager" in the *RTSM Administration Guide*.

The following sections expand on each of the phases showing the most critical steps and best practices:

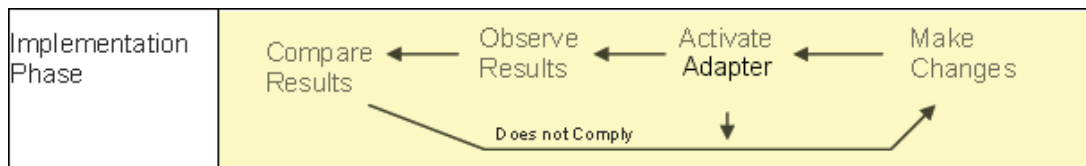
- "Research and Preparation Phase" on the next page
- "Adapter Development and Testing" on the next page
- "Adapter Packaging and Productization " on page 19

Research and Preparation Phase



The Research and Preparation phase encompasses the driving business needs and use cases, and also accounts for securing the necessary facilities to develop and test the adapter.

1. When planning to modify an existing adapter, the first technical step is to make a backup of that adapter and ensure you can return it to its pristine state. If you plan to create a new adapter, copy the most similar adapter and save it under an appropriate name. For details, see Resources Pane in the *Data Flow Management Guide*.
2. Research how the adapter should collect data:
 - Use External tools/protocols to obtain the data
 - Develop how the adapter should create CIs based on the data
 - You now know what a similar adapter should look like
3. Determine most similar adapter based on:
 - Same CIs created
 - Same Protocols used (SNMP)
 - Same kind of targets (by OS type, versions, and so on)
4. Copy entire package.
5. Unzip into work space and rename the adapter (XML) and Jython (.py) files.



Adapter Development and Testing

The Adapter Development and Testing phase is a highly iterative process. As the adapter begins to take shape, you begin testing against the final use cases, make changes, test again, and repeat this process until the adapter complies with the requirements.

Startup and Preparation of Copy

- Modify XML parts of the adapter: Name (id) in line 1, Created CI Types, and Called Jython script name.
- Get the copy running with identical results to the original adapter.
- Comment out most of the code, especially the critical result-producing code.

Development and Testing

- Use other sample code to develop changes
- Test adapter by running it
- Use a dedicated view to validate complex results, search to validate simple results

Adapter Packaging and Productization

The Adapter Packaging and Productization phase accounts for the last phase of development. As a best practice, a final pass should be made to clean up debugging remnants, documents, and comments, to look at security considerations, and so on, before moving on to packaging. You should always have at least a readme document to explain the inner workings of the adapter. Someone (maybe even you) may need to look at this adapter in the future and will be aided greatly by even the most limited documentation.

Cleanup and Document

- Remove debugging
- Comment all functions and add some opening comments in the main section
- Create sample TQL and view for the user to test

Create Package

- Export adapters, TQL, and so on with the Package Manager. For details, see "Package Manager" in the *RTSM Administration Guide*.
- Check any dependencies your package has on other packages, for example, if the CIs created by those packages are input CIs to your adapter.
- Use Package Manager to create a package zip. For details, see "Package Manager" in the *RTSM Administration Guide*.
- Test deployment by removing parts of the new content and redeploying, or deploying on another test system.

Data Flow Management and Integration

DFM adapters are capable of integration with other products. Consider the following definitions:

- DFM collects specific content from many targets.
- Integration collects multiple types of content from one system.

Note that these definitions do not distinguish between the methods of collection. Neither does DFM. The process of developing a new adapter is the same process for developing new integration. You do the same research, make the same choices for new vs. existing adapters, write the adapters the same way, and so on. Only a few things change:

- The final adapter's scheduling. Integration adapters may run more frequently than discovery, but it depends on the use cases.
- Input CIs:
 - Integration: non-CI trigger to run with no input: a file name or source is passed through the adapter parameter.
 - Discovery: uses regular, RTSM CIs for input.

For integration projects, you should almost always reuse an existing adapter. The direction of the integration (from Run-time Service Model to another product, or from another product to Run-time Service Model) may affect your approach to development. There are field packages available for you to copy for your own uses, using proven techniques.

From Run-time Service Model to another project:

- Create a TQL that produces the CIs and relations to be exported.
- Use a generic wrapper adapter to execute the TQL and write the results to an XML file for the external product to read.

Note: For examples of field packages, contact HP Software Support.

To integrate another product to Run-time Service Model, depending on how the other product exposes its data, the integration adapter acts differently:

Integration Type	Reference Example to Be Reused
Access the product's database directly	HP ED
Read in a csv or xml file produced by an export	HP ServiceCenter
Access a product's API	BMC Atrium/Remedy

Associating Business Value with Discovery Development

The use case for developing new discovery content should be driven by a business case and plan to produce business value. That is, the goal of mapping system components to CIs and adding them to the RTSM is to provide business value.

The content may not always be used for application mapping, although this is a common intermediate step for many use cases. Regardless of the end usage of the content, your plan should answer these questions of this approach:

- Who is the consumer? How should the consumer act on the information provided by the CIs (and the relationships between them)? What is the business context in which the CIs and relationships are to be viewed? Is the consumer of these CIs a person or a product or both?
- Once the perfect combination of CIs and relationships exists in the RTSM, how do I plan on using them to produce business value?
- What should the perfect mapping look like?

- What term would most meaningfully describe the relationships between each CI?
- What types of CIs would be most important to include?
- What is the end usage and end user of the map?
- What would be the perfect report layout?

Once the business justification is established, the next step is to embody the business value in a document. This means picturing the perfect map using a drawing tool and understanding the impact and dependencies between CIs, reports, how changes are tracked, what change is important, monitoring, compliance, and additional business value as required by the use cases.

This drawing (or model) is referred as the **blueprint**.

For example, if it is critical for the application to know when a certain configuration file has changed, the file should be mapped and linked to the appropriate CI (to which it relates) in the drawn map.

Work with an SME (Subject Matter Expert) of the area, who is the end user of the developed content. This expert should point out the critical entities (CIs with attributes and relationships) that must exist in the RTSM to provide business value.

One method could be to provide a questionnaire to the application owner (also the SME in this case). The owner should be able to specify the above goals and blueprint. The owner must at least provide a current architecture of the application.

You should map critical data only and no unnecessary data: you can always enhance the adapter later. The goal should be to set up a limited discovery that works and provides value. Mapping large quantities of data gives more impressive maps but can be confusing and time consuming to develop.

Once the model and business value is clear, continue to the next stage. This stage can be revisited as more concrete information is provided from the next stages.

Researching Integration Requirements

The prerequisite of this stage is a **blueprint** of the CIs and relationships needed to be discovered by DFM, which should include the attributes that are to be discovered. For details, see "[Adapter Development and Writing Overview](#)" on page 16.

This section includes the following topics:

- "[Modifying an Existing Adapter](#)" below
- "[Writing a New Adapter](#)" on the next page
- "[Model Research](#)" on the next page
- "[Technology Research](#)" on the next page
- "[Guidelines for Choosing Ways to Access Data](#)" on page 23
- "[Summary](#)" on page 23

Modifying an Existing Adapter

You modify an existing adapter when an out-of-the-box or field adapter exists, but:

- it does not discover specific attributes that are needed
- a specific type of target (OS) is not being discovered or is being incorrectly discovered
- a specific relationship is not being discovered or created

If an existing adapter does some, but not all, of the job, your first approach should be to evaluate the existing adapters and verify if one of them almost does what is needed; if it does, you can modify the existing adapter.

You should also evaluate if an existing field adapter is available. Field adapters are discovery adapters that are available but are not out-of-the-box. Contact HP Software Support to receive the current list of field adapters.

Writing a New Adapter

A new adapter needs to be developed:

- When it is faster to write an adapter than to insert the information manually into the RTSM (generally, from about 50 to 100 CIs and relationships) or it is not a one-time effort.
- When the need justifies the effort.
- If out-of-the-box or field adapters are not available.
- If the results can be reused.
- When the target environment or its data is available (you cannot discover what you cannot see).

Model Research

- Browse the BSM class model (CI Type Manager) and match the entities and relations from your **blueprint** to existing CITs. It is highly recommended to adhere to the current model to avoid possible complications during version upgrade. If you need to extend the model, you should create new CITs since an upgrade may overwrite out-of-the-box CITs.
- If some entities, relations, or attributes are lacking from the current model, you should create them. It is preferable to create a package with these CITs (which will also later hold all the discovery, views, and other artifacts relating to this package) since you need to be able to deploy these CITs on each installation of Run-time Service Model.

Technology Research

Once you have verified that the RTSM holds the relevant CIs, the next stage is to decide how to retrieve this data from the relevant systems.

Retrieving data usually involves using a protocol to access a management part of the application, actual data of the application, or configuration files or databases that are related to the application. Any data source that can provide information on a system is valuable. Technology research requires both extensive knowledge of the system in question and sometimes creativity.

For home-grown applications, it may be helpful to provide a questionnaire form to the application owner. In this form the owner should list all the areas in the application that can provide information needed for the blueprint and business values. This information should include (but does not have to be limited to) management databases, configuration files, log files, management interfaces, administration programs, Web services, messages or events sent, and so on.

For off-the-shelf products, you should focus on documentation, forums, or support of the product. Look for administration guides, plug-ins and integrations guides, management guides, and so on. If

data is still missing from the management interfaces, read about the configuration files of the application, registry entries, log files, NT event logs, and any artifacts of the application that control its correct operation.

Guidelines for Choosing Ways to Access Data

Relevance: Select sources or a combination of sources that provide the most data. If a single source supplies most information whereas the rest of the information is scattered or hard to access, try to assess the value of the remaining information by comparison with the effort or risk of getting it. Sometimes you may decide to reduce the blueprint if the value or cost does not warrant the invested effort.

Reuse: If Run-time Service Model already includes a specific connection protocol support it is a good reason to use it. It means the DFM Framework is able to supply a ready made client and configuration for the connection. Otherwise, you may need to invest in infrastructure development. You can view the currently supported Run-time Service Model connection protocols: **Data Flow Management > Data Flow Probe Setup > Domains and Probes pane**. For details, see Domains and Probes Pane in the *Data Flow Management Guide*.

You can add new protocols by adding new CIs to the model. For details, contact HP Software Support.

Note: To access Windows Registry data, you can use either WMI or NTCMD.

Security: Access to information usually requires credentials (user name, password), which are entered in the RTSM and are kept secure throughout the product. If possible, and if adding security does not conflict with other principles you have set, choose the least sensitive credential or protocol that still answers access needs. For example, if information is available both through JMX (standard administration interface, limited) and Telnet, it is preferable to use JMX since it inherently provides limited access and (usually) no access to the underlying platform.

Comfort: Some management interfaces may include more advanced features. For example, it might be easier to issue queries (SQL, WMI) than to navigate information trees or build regular expressions for parsing.

Developer Audience: The people who will eventually develop adapters may have an inclination towards a certain technology. This can also be considered if two technologies provide almost the same information at an equal cost in other factors.

Summary

The outcome of this stage is a document describing the access methods and the relevant information that can be extracted from each method. The document should also contain a mapping from each source to each relevant blueprint data.

Each access method should be marked according to the above instructions. Finally you should now have a plan of which sources to discover and what information to extract from each source into the blueprint model (which should by now have been mapped to the corresponding BSM model).

Developing Integration Content

Before creating a new integration, you must understand what the integration's requirements are:

- Should the integration copy data into the RTSM? Should the data be tracked by history? Is the source unreliable?

Population is needed.

- Should the integration federate data on the fly for views and TQL queries? Is the accuracy of changes to data critical? Is the amount of data too large to copy to the RTSM, but the requested amount of data is usually small?

Federation is needed.

- Should the integration push data in to remote data sources?

Data Push is needed.

Note: Federation and Population flows may be configured for the same integration, for maximum flexibility.

For details about the different types of integrations, see Integration Studio in the *Data Flow Management Guide*.

Four different options are available for creating integration adapters:

- Jython Adapter
 - The classic discovery pattern
 - Written in Jython
 - Used for population

For details, see ["Developing Jython Adapters" on page 51](#).
- Java Adapter
 - An adapter that implements one of the adapter interfaces in the Federation SDK Framework.
 - May be used for one or more of Federation, Population, or Data Push (depending on the required implementation).
 - Written from scratch in Java, which allows writing code that will connect to any possible source or target.
 - Suitable for jobs that each connect a single data source or target.

For details, see ["Developing Java Adapters" on page 176](#).
- Generic DB Adapter
 - An abstract adapter based on the Java Adapter and uses the Federation SDK Framework).
 - Allows creation of adapters that connect to external data repositories.
 - Supports both Federation and Population (with a Java plugin implemented for changes support).

- Relatively easy to define, as it is based mainly on XML and property configuration files.
- Main configuration is based on an **orm.xml** file that maps between BSM classes and database columns.
- Suitable for jobs that each connect a single data source.

For details, see ["Developing Generic Database Adapters" on page 98](#).

- Generic Push Adapter

- An abstract adapter based on the Java Adapter (the Federation SDK Framework) and the Jython Adapter.
- Allows creation of adapters that push data to remote targets.
- Relatively easy to define, as you need only to define the mapping between BSM classes and XML, and a Jython script that pushes the data to the target.
- Suitable for jobs that each connect a single data target.
- Used for Data Push.

For details, see ["Developing Push Adapters" on page 211](#).

The following table displays the capabilities of each adapter:

Flow/Adapter	Jython Adapter	Java Adapter	GDB Adapter	Push Adapter
Population	X	X	X	
Federation		X	X	
Data Push		X		X

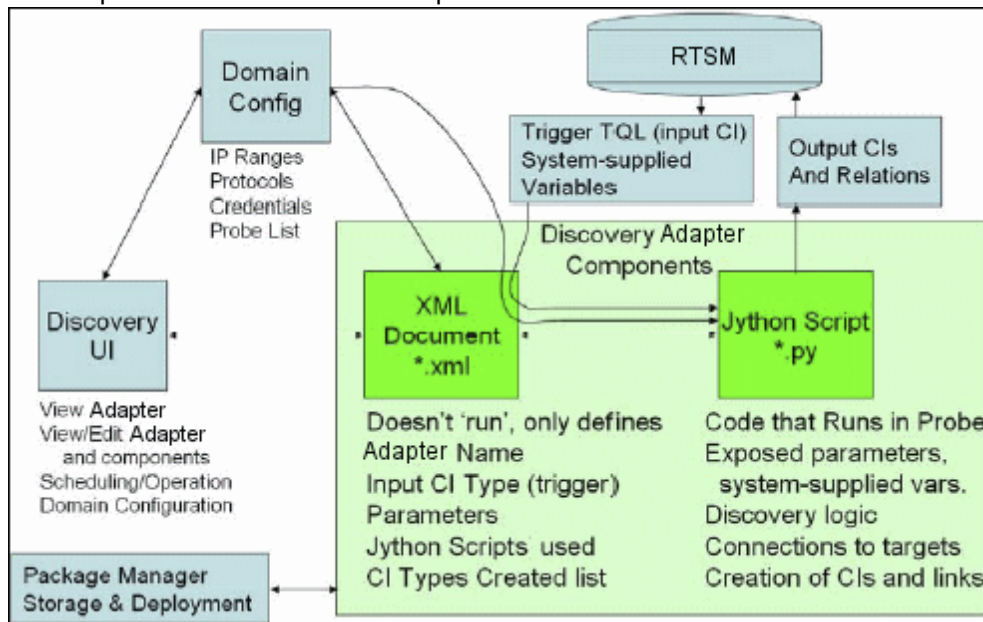
Developing Discovery Content

This section includes:

- "Discovery Adapters and Related Components " below
- "Separating Adapters" on the next page

Discovery Adapters and Related Components

The following diagram shows an adapter's components and the components they interact with to execute discovery. The components in green are the actual adapters, and the components in blue are components that interact with adapters.



Note that the minimum notion of an adapter is two files: an XML document and a Jython script. The Discovery Framework, including input CIs, credentials, and user-supplied libraries, is exposed to the adapter at run time. Both discovery adapter components are administered through Data Flow Management. They are stored operationally in the RTSM itself; although the external package remains, it is not referred to for operation. The Package Manager enables preservation of the new discovery and integration content capability.

Input CIs to the adapter are provided by a TQL, and are exposed to the adapter script in system-supplied variables. Adapter parameters are also supplied as destination data, so you can configure the adapter's operation according to an adapter's specific function.

The DFM application is used to create and test new adapters. You use the Discovery Control Panel, Adapter Management, and Data Flow Probe Setup pages during adapter writing.

Adapters are stored and transported as packages. The Package Manager application and the JMX console are used to create packages from newly created adapters, and to deploy adapters on new systems.

Separating Adapters

Technically, an entire discovery could be defined in a single adapter. But good design demands that a complex system be separated into simpler, more manageable components.

The following are guidelines and best practices for dividing the adapter process:

- Discovery should be done in stages. Each stage should be represented by an adapter that should map an area or tier of the system. Adapters should rely on the previous stage or tier to be discovered, to continue discovery of the system. For example, Adapter A is triggered by an application server TQL result and maps the application server tier. As part of this mapping, a JDBC connection component is mapped. Adapter B registers a JDBC connection component as a trigger TQL and uses the results of adapter A to access the database tier (for example, through the JDBC URL attribute) and maps the database tier.
- **The two-phase connect paradigm:** Most systems require credentials to access their data. This means that a user/password combination needs to be tried against these systems. The DFM administrator supplies credentials information in a secure way to the system and can give several, prioritized login credentials. This is referred to as the **Protocol Dictionary**. If the system is not accessible (for whatever reason) there is no point in performing further discovery. If the connection is successful, there needs to be a way to indicate which credential set was successfully used, for future discovery access.

These two phases lead to a separation of the two adapters in the following cases:

- **Connection Adapter:** This is an adapter that accepts an initial trigger and looks for the existence of a remote agent on that trigger. It does so by trying all entries in the Protocol Dictionary which match this agent's type. If successful, this adapter provides as its result a remote agent CI (SNMP, WMI, and so on), which also points to the correct entry in the Protocol Dictionary for future connections. This agent CI is then part of a trigger for the content adapter.
- **Content Adapter:** This adapter's precondition is the successful connection of the previous adapter (preconditions specified by the TQLs). These types of adapters no longer need to look through all of the Protocol Dictionary since they have a way to obtain the correct credentials from the remote agent CI and use them to log in to the discovered system.
- Different scheduling considerations can also influence discovery division. For example, a system may only be queried during off hours, so even though it would make sense to join the adapter to the same adapter discovering another system, the different schedules mean that you need to create two adapters.
- Discovery of different management interfaces or technologies to discover the same system should be placed in separate adapters. This is so that you can activate the access method appropriate for each system or organization. For example, some organizations have WMI access to machines but do not have SNMP agents installed on them.

Implement a Discovery Adapter

A DFM task has the aim of accessing remote (or local) systems, modeling extracted data as CIs, and saving the CIs to the RTSM. The task consists of the following steps:

1. **Create an adapter.**

You configure an adapter file that holds the context, parameters, and result types by selecting the scripts that are to be part of the adapter. For details, see ["Step 1: Create an Adapter" on page 31](#).

2. **Create a Discovery job.**

You configure a job with scheduling information and a trigger query. For details, see ["Step 2: Assign a Job to the Adapter" on page 38](#).

3. **Edit Discovery code.**

You can edit the Jython or Java code that is contained in the adapter files and that refers to the DFM Framework. For details, see ["Step 3: Create Jython Code" on page 40](#).

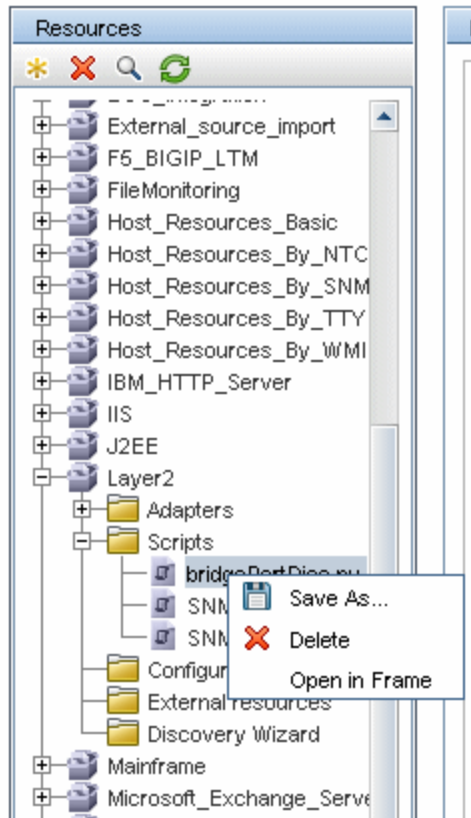
To write new adapters, you create each of the above components, each one of which is automatically bound to the component in the previous step. For example, once you create a job and select the relevant adapter, the adapter file binds to the job.

Adapter Code

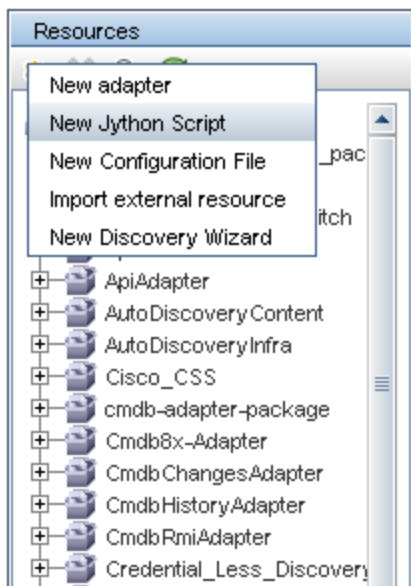
The actual implementation of connecting to the remote system, querying its data, and mapping it as RTSM data is performed by the Jython code. For example, the code contains the logic for connecting to a database and extracting data from it. In this case, the code expects to receive a JDBC URL, a user name, a password, a port, and so on. These parameters are specific for each instance of the database that answers the TQL query. You define these variables in the adapter (in the Trigger CI data) and when the job runs, these specific details are passed to the code for execution.

The adapter can refer to this code by a Java class name or a Jython script name. In this section we discuss writing DFM code as Jython scripts.

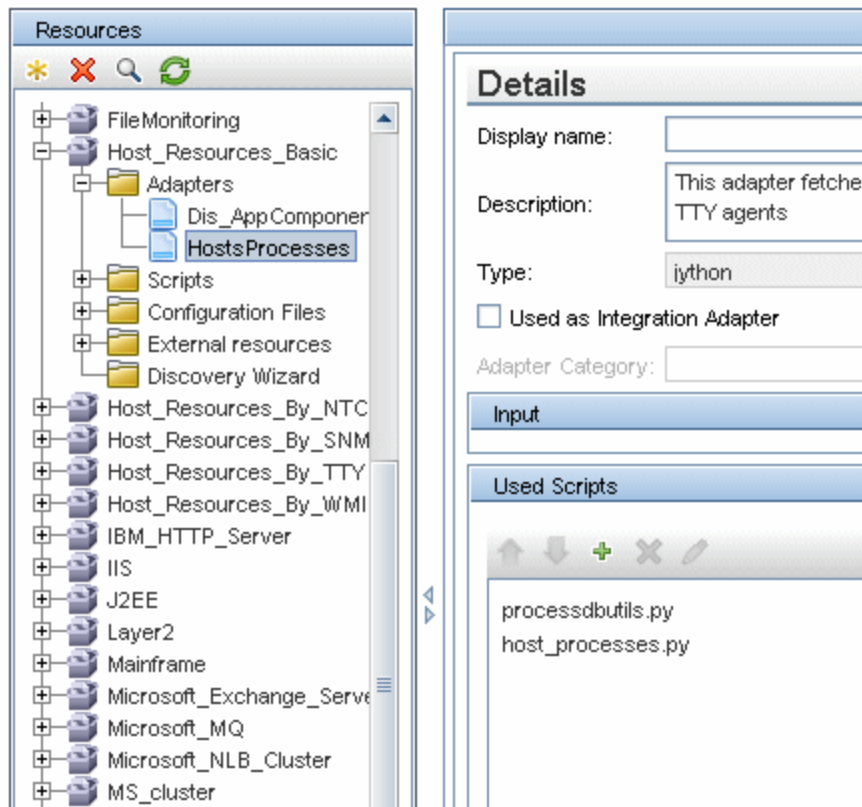
An adapter can contain a list of scripts to be used when running discovery. When creating a new adapter, you usually create a new script and assign it to the adapter. A new script includes basic templates, but you can use one of the other scripts as a template by right-clicking it and selecting **Save as**:



For details on writing new Jython scripts, see "Step 3: Create Jython Code" on page 40. You add scripts through the Resources pane:



The list of scripts are run one after the other, in the order in which they are defined in the adapter:



Note: A script must be specified even though it is being used solely as a library by another script. In this case, the library script must be defined before the script using it. In this example, the `processdbutils.py` script is a library used by the last `host_processes.py` script. Libraries are distinguished from regular runnable scripts by the lack of the `DiscoveryMain()` function.

Step 1: Create an Adapter

An adapter can be considered as the definition of a function. This function defines an input definition, runs logic on the input, defines the output, and provides a result.

Each adapter specifies input and output: Both input and output are Trigger CIs that are specifically defined in the adapter. The adapter extracts data from the input Trigger CI and passes this data as parameters to the code. (Data from related CIs is sometimes passed to the code too. For details, see "Related CIs Window" in the *Data Flow Management Guide*.) An adapter's code is generic, apart from these specific input Trigger CI parameters that are passed to the code.

For details on input components, see "Data Flow Management Concepts" in the *Data Flow Management Guide*.

This section includes the following topics:

- ["Define Adapter Input \(Trigger CIT and Input Query\)" below](#)
- ["Define Adapter Output" on page 34](#)
- ["Override Adapter Parameters" on page 35](#)
- ["Override Probe Selection - Optional" on page 36](#)
- ["Configure a classpath for a remote process - Optional" on page 37](#)

1. Define Adapter Input (Trigger CIT and Input Query)

You use the Trigger CIT and Input Query components to define specific CIs as adapter input:

- The Trigger CIT defines which CIT is used as the input for the adapter. For example, for an adapter that is going to discover IPs, the input CIT is Network.
- The Input query is a regular, editable query that defines the query against the RTSM. The Input Query defines additional constraints on the CIT (for example, if the task requires a `hostID` or `application_ip` attribute), and can define more CI data, if needed by the adapter.

If the adapter requires additional information from the CIs that are related to the Trigger CI, you can add additional nodes to the input TQL. For details, see ["Example of Input Query Definition on the next page below"](#) and "Add Query Nodes and Relationships to a TQL Query" in the Modeling Guide.

- The Trigger CI data contains all the required information on the Trigger CI as well as information from the other nodes in the Input TQL, if they are defined. DFM uses variables to retrieve data from the CIs. When the task is downloaded to the Probe, the Trigger CI data variables are replaced with actual values that exist on the attributes for real CI instances.

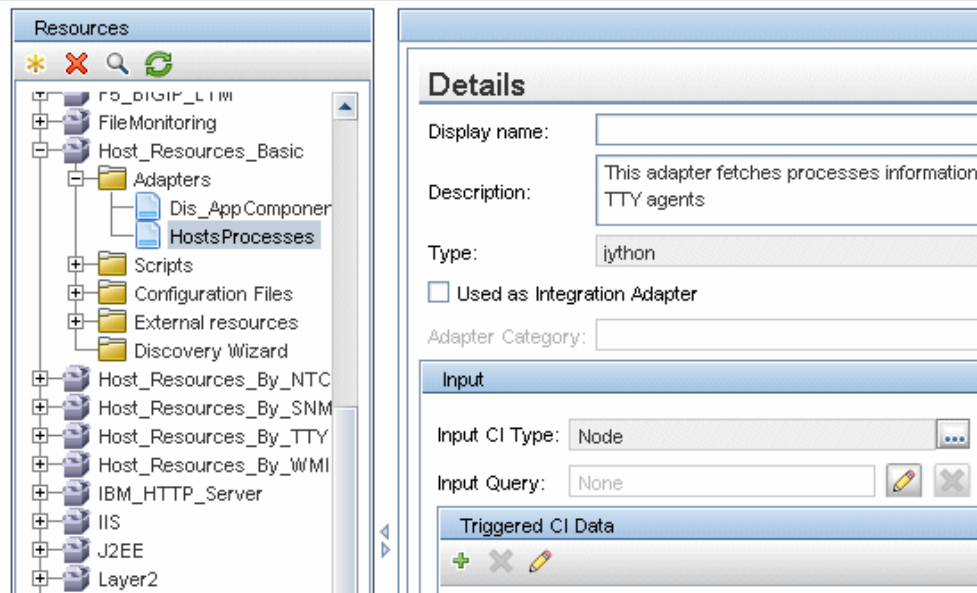
Example of Trigger CIT Definition:

In this example, a Trigger CIT defines that IP CIs are permitted in the adapter.

- a. Access **Admin > RTSM Administration > Data Flow Management > Adapter Management**. Select the **HostProcesses** adapter (**Packages > Host_Resources_Basic > Adapters > HostProcesses**).

- b. Locate the Input CI Type box. For details, see "Adapter Definition Tab" in the *Data Flow Management Guide*.
- c. Click the button to open the Choose Discovered Class dialog box. For details, see "Choose Discovered Class Dialog Box" in the *Data Flow Management Guide*.
- d. Select the CIT.

In this example, the IP CI (Host) is permitted in the adapter:

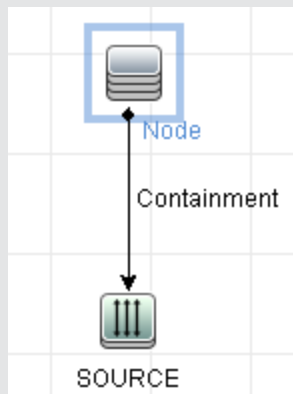


Example of Input Query Definition

In this example, the Input TQL query defines that the `IpAddress` CI (configured in the previous example as the Trigger CIT) must be connected to a `Node` CI.

- a. Access **Admin > RTSM Administration > Data Flow Management > Adapter Management**. Locate the Input TQL box. Click the **Edit** button to open the Input TQL Editor. For details, see "Input Query Editor Window" in the *Data Flow Management Guide*.
- b. In the Input TQL Editor, name the Trigger CI node **SOURCE**: right-click the node and choose **Query Node Properties**. In the **Element Name** box, change the name to **SOURCE**.
- c. Add a `Node` CI and a `Containment` relationship to the `IpAddress` CI. For details on working with the Input TQL Editor, see "Input Query Editor Window" in the *Data Flow*

Management Guide.



The **IpAddress** CI is connected to a **Node** CI. The input TQL consists of two nodes, **Node** and **IpAddress**, with a link between them. The **IpAddress** CI is named **SOURCE**.

Example of Adding Variables to the Input TQL Query:

In this example, you add **DIRECTORY** and **CONFIGURATION_FILE** variables to the Input TQL query created in the previous example. These variables help to define what must be discovered, in this case, to find the configuration files residing on the hosts that are linked to the IPs you need to discover.

- a. Display the Input TQL created in the previous example.

Access **Admin > RTSM Administration > Data Flow Management > Adapter Management**. Locate the Triggered CI Data pane. For details, see "Adapter Definition Tab" in the *Data Flow Management Guide*.

- b. Add variables to the Input TQL. For details, access **Admin > RTSM Administration > Data Flow Management > Adapter Management**. Locate the Triggered CI Data pane. For details, see the Variables field in "Adapter Definition Tab" in the *Data Flow Management Guide*.

The screenshot shows a 'Parameter Editor' dialog box with a title bar containing a small icon and a close button. Inside the dialog, there are three input fields: 'Name' (a single-line text box), 'Value' (a single-line text box), and 'Description' (a multi-line text area). At the bottom right of the dialog, there are two buttons: 'OK' and 'Cancel'.

Example of Replacing Variables with Actual Data:

In this example, variables replace the **IpAddress** CI data with actual values that exist on real **IpAddress** CI instances in your system.

The Triggered CI data for the **IpAddress** CI includes a `fileName` variable. This variable enables the replacement of the **CONFIGURATION_DOCUMENT** node in the Input TQL with the actual values of the configuration file located on a host:

Triggered CI data	
<div> <div>+</div> <div>✖</div> <div>✎</div> </div>	
Name	Value
Protocol	\${SOURCE.credentials_id}
credentialsId	\${SOURCE.credentials_id}
fileName	\${CONFIGURATION_DOCUMENT.name}
hostID	\${HOST.root_id}
ip_address	\${SOURCE.ip_address}
path	\${CONFIGURATION_DOCUMENT.resource_path}

The Trigger CI data is uploaded to the Probe with all variables replaced by actual values. The adapter script includes a command to use the [DFM Framework](#) to retrieve the actual values of the defined variables:

```
Framework.getTriggerCIData ('ip_address')
```

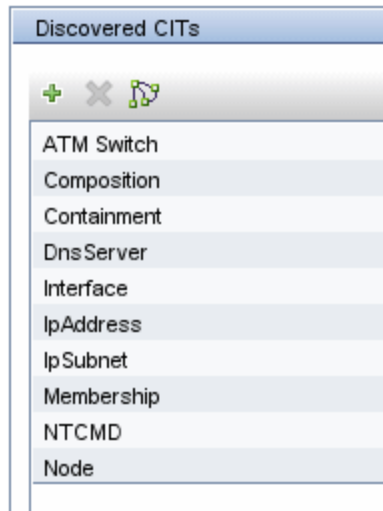
The `fileName` and `path` variables use the `data_name` and `document_path` attributes from the **CONFIGURATION_DOCUMENT** node (defined in the Input Query Editor – see previous example).

The `Protocol`, `credentialsId`, and `ip_address` variables use the `root_class`, `credentials_id`, and `application_ip` attributes:

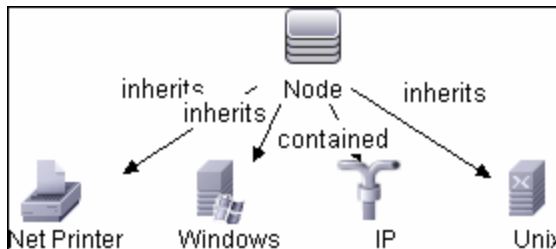
<div> <div>+</div> <div>✎</div> <div>✖</div> <div>↺</div> <div>🔍</div> </div>						
Key	Name	Display Name	Type	Description	Default Value	Visible
	ack_cleared_time	ack_cleared_time	long			
	ack_id	ack_id	string			
🔑	BODY_ICON	BODY_ICON	string		host	
	city	City	string	City location		✓
	codepage	CodePage	string	System su...		
	contextmenu	Context Menu	string_list	Context me... itCIs		
	country	Country	string	Country loc...		✓
	credentials_id	Reference to the cre...	string	Reference ...		
	data_adminstate	Admin State	adminstate...	Admin State	Managed	

2. Define Adapter Output

The output of the adapter is a list of discovered CIs (**Admin > RTSM Administration > Data Flow Management > Adapter Management > Adapter Definition** tab > **Discovered CITs**) and the links between them:



You can also view the CITs as a topology map, that is, the components and the way in which they are linked together (click the **View Discovered CITs as Map** button):



The discovered CIs are returned by the DFM code (that is, the Jython script) in the format of BSM's `ObjectStateHolderVector`. For details, see ["Results Generation by the Jython Script" on page 57](#).

Example of Adapter Output:

In this example, you define which CITs are to be part of the IP CI output.

- Access **Admin > RTSM Administration > Data Flow Management > Adapter Management**.
- In the Resources pane, select **Network > Adapters > NSLOOKUP_on_Probe**.
- In the Adapter Definition tab, locate the Discovered CITs pane.
- The CITs that are to be part of the adapter output are listed. Add CITs to, or remove from, the list. For details, see *"Adapter Definition Tab" in the Data Flow Management Guide*.

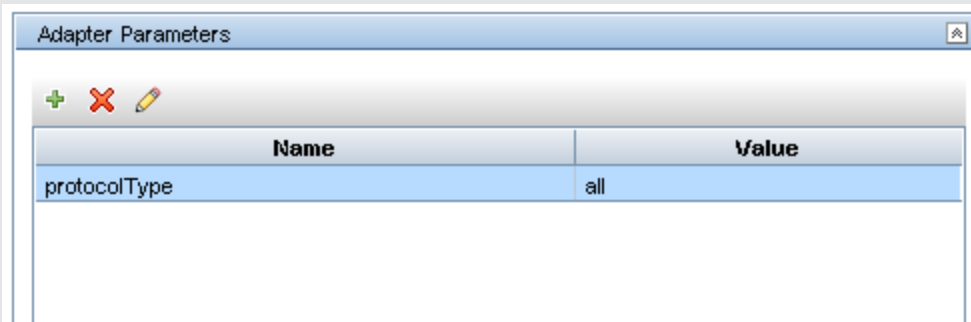
3. Override Adapter Parameters

To configure an adapter for more than one job, you can override adapter parameters. For example, the adapter `SQL_NET_Dis_Connection` is used by both the `MSSQL Connection by SQL` and the `Oracle Connection by SQL` jobs.

Example of Overriding an Adapter Parameter:

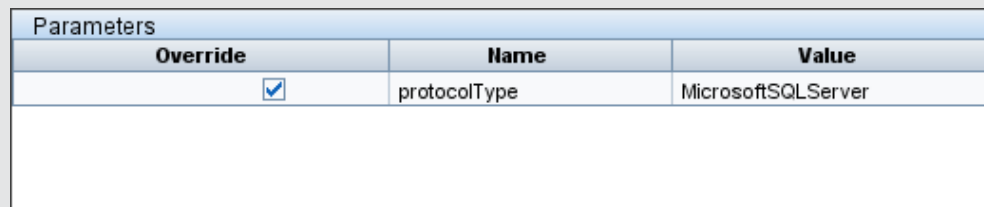
This example illustrates overriding an adapter parameter so that one adapter can be used to discover both Microsoft SQL Server and Oracle databases.

- Access **Admin > RTSM Administration > Data Flow Management > Adapter Management**.
- In the Resources pane, select **Database Basic > Adapters > SQL_NET_Dis_Connection**.
- In the Adapter Definition tab, locate the **Discovery Pattern Parameters** pane. The `protocolType` parameter has a value of `all`:



Name	Value
protocolType	all

- Right-click the **SQL_NET_Dis_Connection_MsSql** adapter and choose **Go to Discovery Job > MSSQL Connection by SQL**.
- Display the Properties tab. Locate the Parameters pane:



Override	Name	Value
<input checked="" type="checkbox"/>	protocolType	MicrosoftSQLServer

The `all` value is overwritten with the `MicrosoftSQLServer` value.

Note: The **Oracle Connection by SQL** job includes the same parameter but the value is overwritten with an Oracle value.

For details on adding, deleting, or editing parameters, see "Adapter Definition Tab" in the *Data Flow Management Guide*.

DFM begins looking for Microsoft SQL Server instances according to this parameter.

4. Override Probe Selection - Optional

In the UCMDDB server there is a dispatching mechanism that takes the trigger CIs received by the UCMDDB and automatically chooses which probe should run the job for each trigger CI according to one of the following options.

- **For the IP address CI type:** take the probe that is defined for this IP.
- **For the running software CI type:** use the attributes `application_ip` and `application_ip_`

domain and choose the probe that is defined for the IP in the relevant domain.

- **For other CI types:** take the node's IP according to the CI's related node (if it exists).

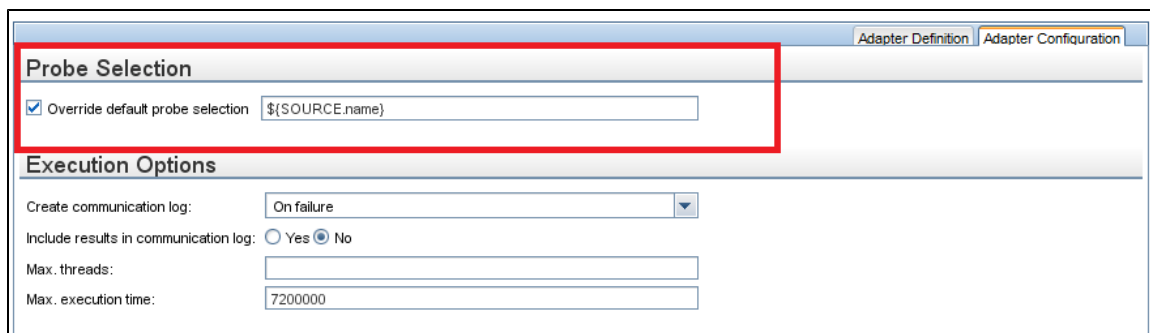
The automatic probe selection is done according to the CI's related node. After obtaining the CI's related node, the dispatching mechanism chooses one of the node's IPs and chooses the probe according to the probe's network scope definitions.

In the following cases, you need to specify the probe manually and not use the automatic dispatching mechanism:

- You already know which probe should be run for the adapter and you do not need the automatic dispatching mechanism to select the probe (for example if the trigger CI is the probe gateway).
- The automatic probe selection might fail. This can happen in the following situations:
 - A trigger CI does not have a related node (such as the network CIT)
 - A trigger CI's node has multiple IPs, each belonging to a different probe.

To resolve these issues, you can specify which probe to use with the adapter as follows:

- a. In the Probe selection section, select **Override default probe selection** as shown below.



The screenshot shows the 'Adapter Configuration' tab in a software interface. The 'Probe Selection' section is highlighted with a red rectangle. It contains a checkbox labeled 'Override default probe selection' which is checked, followed by a text input field containing the value '\$(SOURCE.name)'. Below this, the 'Execution Options' section is visible, containing several configuration fields: 'Create communication log' set to 'On failure', 'Include results in communication log' with 'Yes' and 'No' radio buttons (where 'No' is selected), 'Max. threads' with an empty input field, and 'Max. execution time' set to '7200000'.

- b. In the Probe box, type the probe to use for the task.

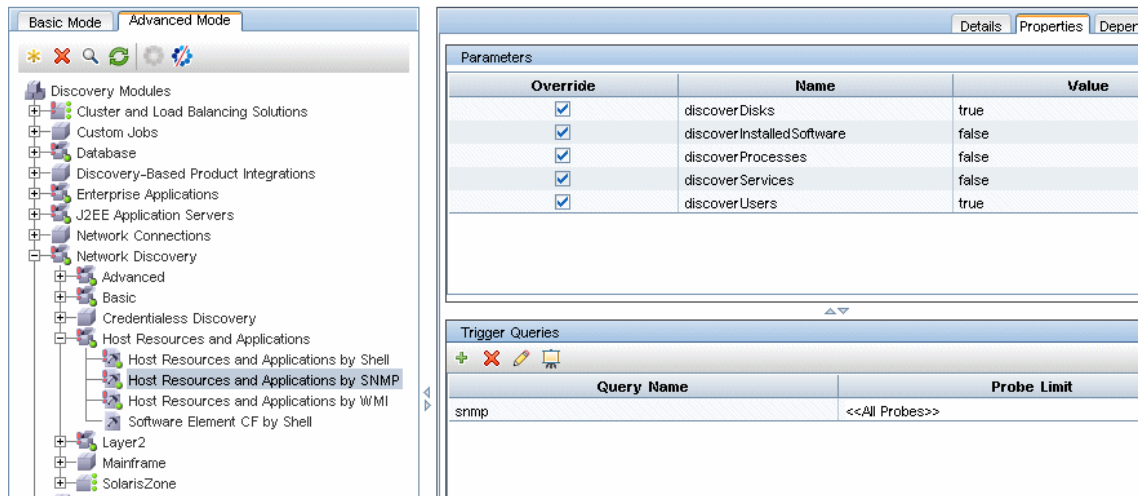
5. Configure a classpath for a remote process - Optional

For details, see ["Configure Remote Process Execution" on page 41](#).

Step 2: Assign a Job to the Adapter

Each adapter has one or more associated jobs that define the execution policy. Jobs enable scheduling the same adapter differently over different sets of Triggered CIs and also enable supplying different parameters for each set.

The jobs appear in the Discovery Modules tree, and this is the entity that the user activates, as shown in the picture below.



Choose a Trigger TQL

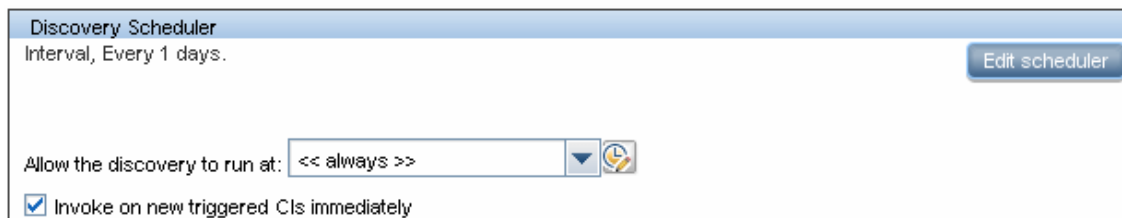
Each job is associated with Trigger TQLs. These Trigger TQLs publish results that are used as Input Trigger CIs for the adapter of this job.

A Trigger TQL can add constraints to an Input TQL. For example, if an input TQL's results are IPs connected to SNMP, a trigger TQL's results can be IPs connected to SNMP within the range 195.0.0.0-195.0.0.10.

Note: A trigger TQL must refer to the same objects that the input TQL refers to. For example, if an input TQL queries for IPs running SNMP, you cannot define a trigger TQL (for the same job) to query for IPs connected to a host, because some of the IPs may not be connected to an SNMP object, as required by the input TQL.

Set Scheduling Information

The scheduling information for the Probe specifies when to run the code on Trigger CIs. If the **Invoke on new triggered CIs Immediately** check box is selected, the code also runs once on each Trigger CI when it reaches the Probe, regardless of future schedule settings.



For each scheduled occurrence for each job, the Probe runs the code against all Trigger CIs accumulated for that job. For details, see Discovery Scheduler Dialog Box in the *Data Flow Management Guide*.

Override Adapter Parameters

When configuring a job you can override the adapter parameters. For details, see ["Override Adapter Parameters" on page 35](#).

Step 3: Create Jython Code

Run-time Service Model uses Jython scripts for adapter-writing. For example, the `SNMP_Connection.py` script is used by the `SNMP_NET_Dis_Connection` adapter to try and connect to machines using SNMP. Jython is a language based on Python and powered by Java.

For details on how to work in Jython, you can refer to these Web sites:

- <http://www.jython.org>
- <http://www.python.org>

For details, see "[Create Jython Code](#)" on page 53.

Configure Remote Process Execution

You can run discovery for a discovery job in a process separate from the Data Flow Probe's process.

For example, you can run the job in a separate remote process if the job uses jar libraries that are a different version to the Probe's libraries or that are incompatible with the Probe's libraries.

You can also run the job in a separate remote process if the job potentially consumes a lot of memory (brings a lot of data) and you want to isolate the Probe from potential OutOMemory problems.

To configure a job to run as a remote process, define the following parameters in its adapter's configuration file:

Parameter	Description
remoteJVMArgs	JVM parameters for the remote Java process.
runInSeparateProcess	When set to true , the discovery job runs in a separate process.
remoteJVMClasspath	<p>(Optional) Enables customization of the classpath of the remote process, overriding the default Probe classpath. This is useful if there might be version incompatibility between the Probe's jars and custom jars required for the customer-defined discovery.</p> <p>If the remoteJVMClasspath parameter is not defined, or is left empty, the default Probe classpath is used.</p> <p>If you develop a new discovery job and you want to ensure that the Probe jar library version does not collide with the job's jar libraries, you must use at least the minimal classpath required to execute basic discovery. The minimal classpath is defined in the DiscoveryProbe.properties file in the basic_discovery_minimal_classpath parameter.</p> <p>Examples of remoteJVMClasspath customization:</p> <ul style="list-style-type: none">• To prepend or append custom jars to the default Probe classpath, customize the remoteJVMClasspath parameter as follows: <code>custom1.jar;%classpath%;custom2.jar -</code> In this case, custom1.jar is placed before default Probe classpath, and custom2.jar is appended to the Probe classpath.• To use the minimal classpath, customize the remoteJVMClasspath parameter as follows: <code>custom1.jar;%minimal_classpath%;custom2.jar</code>

Chapter 2

Discovery Content Migration Guidelines

This chapter includes:

- Discovery Content Migration Guidelines Overview43
- Version 9.x New Infrastructure Features 44
- Guidelines for Developing Cross-Data Model Scripts 47
- Implementation Tips48
- Package Migration Utility 49
- Troubleshooting and Limitations50

Discovery Content Migration Guidelines Overview

In BSM version 9.x, the data model significantly evolved, forcing correlated changes in the former Data Flow Management content code. Consequently, some core mechanisms of the Data Flow Management content have changed. Thus, content developed for BSM prior to version 9.x has to be upgraded to correspond with the 9.x data model (UDM: Universal Data Model). This section guides you through the process of adopting Data Flow Management content and aligning it with UDM.

Version 9.x New Infrastructure Features

Note: For details on accessing the UDM documentation online, see [Access Universal Data Model \(UDM\) Documentation Online](#).

This section includes:

Differences between BSM 8.x Class Model and BSM 9.x Data Model

Changes made between the BSM version 8.x class model and UDM are downloaded to the Probe in the following Discovery configuration file:

C:\hp\UCMDB\DataFlowProbe\runtime\probeManager\discoveryConfigFiles\flat-class-model-changes.xml

bdm_changes.xml. This XML file holds information regarding changes made to class names, attributes names, removed classes, attributes, qualifiers, and so on.

New CIT Identification Mechanism

In BSM versions prior to version 9.x, key attributes are used to identify CIs. In BSM version 9.x, this concept has been generalized and the identification is now done in a server component named Reconciliation Engine. The Reconciliation Engine is capable of identifying CIs by logical rules called DDA (Data Definition Algorithm) rules.

This new mechanism is mostly useful for CITs where the related topology is important for their identification (for example, the Node CIT—Host in prior versions—is identified by its name and the related topology, such as the IP Address and Interface CITs). Some CITs are still identified by key attributes; for those CITs, a DDA rule is not defined.

For details about the Reconciliation Engine, see Reconciliation Overview in the Data Flow Management Guide.

Running Software Mechanism

The version 8.x **Software Element** CI is called **Running Software** in version 9.x UDM. This CIT is identified in version 9.x by a DDA rule and not by key attributes.

Say you have added a custom CIT derived from the **Running Software** CIT. In previous versions this custom CIT was identified by its key attributes. However, in version 9.x it is identified by an inherited DDA rule, and thus defined key attributes are ignored.

So if you add a derived CIT, consider the following:

- To identify the new CIT by the same DDA rule as all the Running Software CITs, you should keep the current configuration.
- To identify the new CIT by key attributes, you should create a new DDA rule, defining the identification by key attributes. Following is an example for such a DDA rule, defined for the **object** CIT:

```
<identification-config type="object">
  <identification-criteria>
    <identification-criterion targetType="root">
      <key-attributes-condition>
```

```
        </identification-criterion>
    </identification-criteria>
</identification-config>
```

Probe Side Identification

DDM_ID_ATTRIBUTE. The version 9.x Data Flow Probe identifies CIs only by their key attributes (that is, **ID_ATTRIBUTE**). If a CIT includes a DDA rule (that is, a reconciliation rule), the CIT may not include a key attribute. In this case, the CIT main attributes are marked with a **DDM_ID_ATTRIBUTE** qualifier. Therefore, for the purposes of identifying a CI, the Probe considers all **DDM_ID_ATTRIBUTE** as well as **ID_ATTRIBUTE** qualifiers.

DDM_REQUIRED_TOPOLOGY. A DDA rule for a specific CIT may depend on different CIs reported in the same bulk, together with the examined CI. For example, J2EE Domain CIT identification is carried out not only by the domain name attribute but also by the J2EE Application Server CIT connected to it with a membership link.

To ensure that all the required CIs are reported with the examined CI, you should mark each one of the examined CIs with the **DDM_REQUIRED_TOPOLOGY** qualifier that contains a data item specifying the required link type. For example, in the above example, the J2EE Domain CIT is marked with the **DDM_REQUIRED_TOPOLOGY** qualifier and with a member link data item, so that when Discovery reports a J2EE domain, the servers are also reported. Data item name which specifies link types is **LINK_TYPES**.

As an example, to identify the Node CIT by interfaces and IPs connected to it, then the following qualifier should be added to the Node CIT definition:

```
<Class-Qualifier name="DDM_REQUIRED_TOPOLOGY">
    <Data-Items>
        <Data-Item name="LINK_TYPES" type="string">containment,
composition</Data-Item>
        <Data-Item name="LINK_ENDS" type="string">ip_address,
interface</Data-Item>
        <Data-Item name="LINK_DIRECTIONS" type="string">OUT,OUT</Data-
Item>
        <Data-Item name="APPLY_TO_CHILD_TYPES"
type="string">true</Data-Item>
    </Data-Items>
</Class-Qualifier>
```

where:

- **LINK_TYPES** (mandatory) indicates the link types of the current CIT topology.
- **LINK_ENDS** (optional) provides definitions for the CITs on the opposite ends of the specified link types according to their appearance in the **LINK_TYPES** list. These "opposite" ends are always applied hierarchically.

Omitting the **LINK_ENDS** data item or leaving one end as an empty string in the list means that the opposite end can be of any CIT.

- **LINK_DIRECTION** (optional) indicates the link direction, "OUT", "IN" or "BOTH", to check from the current CIT.

Omitting the LINK_DIRECTION data item or leaving an empty entry in the list means that both directions are checked.

- APPLY_TO_CHILD_TYPES (optional) indicates that the qualifier will be applied recursively to all children of the current CIT.

Omitting the APPLY_TO_CHILD_TYPES data item means that the qualifier is applied only to the current CIT.

If DDM_REQUIRED_TOPOLOGY is defined for a specific CIT, this will override the qualifier defined for its parent.

For details on qualifiers, see Qualifiers Page in the Modeling Guide.

Transformation Layer

To ensure backward compatibility, a new transformation mechanism is introduced in version 9.x on the Probe. The new mechanism is capable of converting version 8.x topologies to 9.x topologies at runtime. It enables the Probe to continue running tasks, such as Jython scripts, which report topologies compatible with version 8.x.

The new transformation mechanism uses the data kept in the **bdm_changes.xml** file, and performs the required changes (class and attributes name changes, attribute removal, hierarchy changes, and so on) to make the 8.x topologies compatible with the UDM. Concurrently (and independently of the topologies reported by the tasks executed by the Probe), the BSM Server receives topologies compatible with UDM.

Guidelines for Developing Cross-Data Model Scripts

The following guidelines are applicable for both version 8.x and 9.x.

Discovery Scripts API Library

The Discovery API library is fully backward compatible and therefore all version 8.x libraries and APIs are supported. For details, see ["Jython Libraries and Utilities" on page 89](#).

The 9.x API includes more elements and methods. For example, a Jython script now reports an error code (integer) instead of a string error message, thus enabling localized discovery error messages. For details, see ["Error-Writing Conventions" on page 94](#).

Implementation Tips

- Use the **modeling** module for creating a **Running Software** CIT or any descendant for which the relevant method is present.
- Use **HostBuilder** for creating CIT of type **Node**.
- Use the **modeling.createOshByCmdbldString** to restore OSH by its ID.
- Use the **ShellUtils** instance of the **shellutils** module for all shell-based connections.
- Use the built-in mechanism to retrieve the BSM version: `logger.Version().getVersion(framework)`. For example, if an additional attribute `application_ip` is added only for BSM version 9.0x or later:

```
versionAsDouble = logger.Version().getVersion(Framework)
if versionAsDouble >= 9:
    appServerOSH.setAttribute('application_ip', ip)
```

- Use **wmiutils** for creating a WMI-based discovery.
- Use **snmputils** for creating a SNMP-based discovery.

Package Migration Utility

The BSM 9.x installation includes an external Package Migration Utility that enables content developers to convert a content package from the 8.x class model to the 9.x data model. The Package Migration Utility converts package resources, subsystem by subsystem, so that they are compatible with the new class model. CIT definitions, queries, jobs, adapters, and modules are transformed according to the data held in the **bdm_changes.xml** file. As a result, they can be deployed and used by a BSM 9.x Server.

Package Migration Utility Limitations

- Jython scripts are not upgraded by the Package Migration Utility. For supporting scripts that are designed to correspond with the BSM version 8.x class model, a new **Transformation layer** module is introduced in BSM 9.x. For details, see "[Transformation Layer](#)" on page 46.
- Discovery Adapters of type Integration are not upgraded by the Package Migration Utility and thus should be upgraded manually.
- The Layer 2 Topology discovery job (and its corresponding resources, such as Discovery Adapter, TQL, and so on) has significantly changed and is removed by the Package Migration Utility instead of being upgraded.

Troubleshooting and Limitations

- The **ip_address** value is not passed by default to the pattern. It should be added explicitly to the pattern as Trigger CI Data.
- If a non-out-of-the-box Jython script requires an external jar or resource in the classpath, it should be located in the relevant package under a sub-folder named **discoveryResources**.
- While working with attributes of type **List** such as **StringVector** and **IntegerVector** (inherited from **BaseVector**), you cannot use both the **add element** and **remove element** operations on the same list object.

Chapter 3

Developing Jython Adapters

This chapter includes:

- HP Data Flow Management API Reference52
- Create Jython Code 53
- Support Localization in Jython Adapters64
- Work with Discovery Analyzer 71
- Run Discovery Analyzer from Eclipse 77
- Record DFM Code87
- Jython Libraries and Utilities89

HP Data Flow Management API Reference

For full documentation on the available APIs, see *HP Universal CMDB Data Flow Management API Reference*. These files are located in the following folder:

\\< Gateway Server root directory>\AppServer\webapps\site.war\amdocs\eng\API_docs\DDM_JavaDoc\index.html

Create Jython Code

Run-time Service Model uses Jython scripts for adapter-writing. For example, the `SNMP_Connection.py` script is used by the `SNMP_NET_Dis_Connection` adapter to try and connect to machines using SNMP. Jython is a language based on Python and powered by Java.

For details on how to work in Jython, you can refer to these Web sites:

- <http://www.jython.org>
- <http://www.python.org>

The following section describes the actual writing of Jython code inside the DFM Framework. This section specifically addresses those contact points between the Jython script and the Framework that it calls, and also describes the Jython libraries and utilities that should be used whenever possible.

Note:

- Scripts written for DFM should be compatible with Jython version 2.1.
- For full documentation on the available APIs, see the *HP Universal CMDB Data Flow Management API Reference*.

This section includes the following topics:

- "Use External Java JAR Files within Jython" below
- "Execution of the Code" on the next page
- "Modifying Out-of-the-Box Scripts" on the next page
- "Structure of the Jython File" on the next page
- "Results Generation by the Jython Script" on page 57
- "The Framework Instance" on page 59
- "Finding the Correct Credentials (for Connection Adapters)" on page 62
- "Handling Exceptions from Java" on page 63

Use External Java JAR Files within Jython

When developing new Jython scripts, external Java Libraries (JAR files) or third-party executable files are sometimes needed as either Java utility archives, connection archives such as JDBC Driver JAR files, or executable files (for example, **nmap.exe** is used for credential-less discovery).

These resources should be bundled in the package under the **External Resources** folder. Any resource put in this folder is automatically sent to any Probe that connects to your RTSM server.

In addition, when discovery is launched, any JAR file resource is loaded into the Jython's classpath, making all the classes within it available for import and use.

Execution of the Code

After a job is activated, a task with all the required information is downloaded to the Probe.

The Probe starts running the DFM code using the information specified in the task.

The Jython code flow starts running from a main entry in the script, executes code to discover CIs, and provides results of a vector of discovered CIs.

Modifying Out-of-the-Box Scripts

When making out-of-the-box script modifications, make only minimal changes to the script and place any necessary methods in an external script. You can track changes more efficiently and, when moving to a newer Run-time Service Model version, your code is not overwritten.

For example, the following single line of code in an out-of-the-box script calls a method that calculates a Web server name in an application-specific way:

```
serverName = iplanet_cspecific.PlugInProcessing(serverName,  
transportHN, mam_utils)
```

The more complex logic that decides how to calculate this name is contained in an external script:

```
# implement customer specific processing for 'servername' attribute of  
httpplugin  
#  
def PlugInProcessing(servername, transportHN, mam_utils_handle):  
    # support application-specific HTTP plug-in naming  
    if servername == "appsrv_instance":  
        # servername is supposed to match up with the j2ee  
server name, however some groups do strange things with their  
        # iPlanet plug-in files. this is the best work-around  
we could find. this join can't be done with IP address:port  
        # because multiple apps on a web server share the same  
IP:port for multiple websphere applications  
        logger.debug('httpcontext_webapplicationserver  
attribute has been changed from [' + servername + '] to [' +  
transportHN[:5] + '] to facilitate websphere enrichment')  
        servername = transportHN[:5]  
    return servername
```

Save the external script in the External Resources folder. For details, see Resources Pane in the Data Flow Management Guide. If you add this script to a package, you can use this script for other jobs, too. For details on working with Package Manager, see "Package Manager" in the *RTSM Administration Guide*.

During upgrade, the change you make to the single line of code is overwritten by the new version of the out-of-the-box script, so you will need to replace the line. However, the external script is not overwritten.

Structure of the Jython File

The Jython file is composed of three parts in a specific order:

1. ["Imports" below](#)
2. ["Structure of the Jython File" on the previous page](#)
3. ["Functions Definition" on the next page](#) (optional)

The following is an example of a Jython script:

```
# imports section
from appilog.common.system.types import ObjectStateHolder
from appilog.common.system.types.vectors import
ObjectStateHolderVector
# Function definition
def foo:
    # do something
# Main Function
def DiscoveryMain(Framework):
    OSHVResult = ObjectStateHolderVector()
    ## Write implementation to return new result CIs here...
    return OSHVResult
```

Imports

Jython classes are spread across hierarchical namespaces. In version 7.0 or later, unlike in previous versions, there are no implicit imports, and so every class you use must be imported explicitly. (This change was made for performance reasons and to enable an easier understanding of the Jython script by not hiding necessary details.)

- To import a Jython script:

```
import logger
```

- To import a Java class:

```
from appilog.collectors.clients import ClientsConsts
```

Main Function – DiscoveryMain

Each Jython runnable script file contains a main function: [DiscoveryMain](#).

The `DiscoveryMain` function is the main entry into the script; it is the first function that runs. The main function may call other functions that are defined in the scripts:

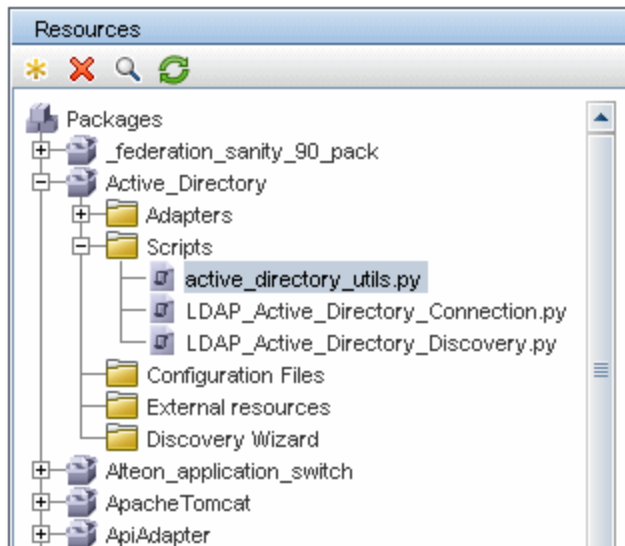
```
def DiscoveryMain(Framework):
```

The `Framework` argument must be specified in the main function definition. This argument is used by the main function to retrieve information that is required to run the scripts (such as information on the Trigger CI and parameters) and can also be used to report on errors that occur during the script run.

You can create a Jython script without any main method. Such scripts are used as library scripts that are called from other scripts.

Functions Definition

Each script can contain additional functions that are called from the main code. Each such function can call another function, which either exists in the current script or in another script (use the `import` statement). Note that to use another script, you must add it to the Scripts section of the package:



Example of a Function Calling Another Function:

In the following example, the main code calls the `doQueryOSUsers(..)` method which calls an internal method `doOSUserOSH(..)`:

```
def doOSUserOSH(name):
    sw_obj = ObjectStateHolder('winosuser')

    sw_obj.setAttribute('data_name', name)
    # return the object
    return sw_obj

def doQueryOSUsers(client, OSHVResult):
    _hostObj = modeling.createHostOSH(client.getIpAddress())
    data_name_mib = '1.3.6.1.4.1.77.1.2.25.1.1,
1.3.6.1.4.1.77.1.2.25.1.2,string'
    resultSet = client.executeQuery(data_name_mib)
    while resultSet.next():
        UserName = resultSet.getString(2)
        ##### send object #####
        OSUserOSH = doOSUserOSH(UserName)
        OSUserOSH.setContainer(_hostObj)
        OSHVResult.add(OSUserOSH)

def DiscoveryMain(Framework):
    OSHVResult = ObjectStateHolderVector()
    try:
```



```
        client = Framework.getClientFactory(ClientsConsts.SNMP_
PROTOCOL_NAME).createClient()
    except:
        Framework.reportError('Connection failed')
    else:
        doQueryOSUsers(client, OSHVResult)
        client.close()
    return OSHVResult
```

If this script is a global library that is relevant to many adapters, you can add it to the list of scripts in the `jythonGlobalLibs.xml` configuration file, instead of adding it to each adapter (**Admin > RTSM Administration > Adapter Management > Resources Pane > AutoDiscoveryContent > Configuration Files**).

Results Generation by the Jython Script

Each Jython script runs on a specific Trigger CI, and ends with results that are returned by the return value of the `DiscoveryMain` function.

The script result is actually a group of CIs and links that are to be inserted or updated in the RTSM. The script returns this group of CIs and links in the format of `ObjectStateHolderVector`.

The `ObjectStateHolder` class is a way to represent an object or link defined in the RTSM. The `ObjectStateHolder` object contains the CIT name and a list of attributes and their values. The `ObjectStateHolderVector` is a vector of `ObjectStateHolder` instances.

The ObjectStateHolder Syntax

This section explains how to build the DFM results into a BSM model.

Example of Setting Attributes on the CIs:

The `ObjectStateHolder` class describes the DFM result graph. Each CI and link (relationship) is placed inside an instance of the `ObjectStateHolder` class as in the following Jython code sample:

```
# siebel application server 1 appServerOSH = ObjectStateHolder('siebelappserver' ) 2
appServerOSH.setStringAttribute('data_name', sbIsvrName) 3
appServerOSH.setStringAttribute('application_ip', ip) 4 appServerOSH.setContainer
(appServerHostOSH)
```

- Line 1 creates a CI of type **siebelappserver**.
- Line 2 creates an attribute called **data_name** with a value of **sbIsvrName** which is a Jython variable set with the value discovered for the server name.
- Line 3 sets a non-key attribute that is updated in the RTSM.
- Line 4 is the building of containment (the result is a graph). It specifies that this application server is contained inside a host (another `ObjectStateHolder` class in the scope).

Note: Each CI being reported by the Jython script must include values for all the key attributes of the CI's CI Type.

Example of Relationships (Links):

The following link example explains how the graph is represented:

```
1 linkOSH = ObjectStateHolder('route') 2 linkOSH.setAttribute('link_end1', gatewayOSH) 3
linkOSH.setAttribute('link_end2', appServerOSH)
```

- Line 1 creates the link (that is also of the `ObjectStateHolder` class. The only difference is that `route` is a link CI Type).
- Lines 2 and 3 specify the nodes at the end of each link. This is done using the **end1** and **end2** attributes of the link which must be specified (because they are the minimal key attributes of each link). The attribute values are `ObjectStateHolder` instances. For details on End 1 and End 2, see Link in the *Data Flow Management Guide*.

Caution: A link is directional. You should verify that End 1 and End 2 nodes correspond to valid CITs at each end. If the nodes are not valid, the result object fails validation and is not reported correctly. For details, see CI Type Relationships in the *HP Universal CMDB Modeling Guide*.

Example of Vector (Gathering CIs):

After creating objects with attributes, and links with objects at their ends, you must now group them together. You do this by adding them to an `ObjectStateHolderVector` instance, as follows:

```
oshvMyResult = ObjectStateHolderVector()
oshvMyResult.add(appServerOSH)
oshvMyResult.add(linkOSH)
```

For details on reporting this composite result to the Framework so it can be sent to the RTSM server, see the `sendObjects` method in the Interface BaseFramework API documentation.

Once the result graph is assembled in an `ObjectStateHolderVector` instance, it must be returned to the DFM Framework to be inserted into the RTSM. This is done by returning the `ObjectStateHolderVector` instance as the result of the `DiscoveryMain()` function.

Note: For details on creating **OSH** for common CITs, see "modeling.py" in "[Jython Libraries and Utilities](#)" on page 89.

The Framework Instance

The Framework instance is the only argument that is supplied in the main function in the Jython script. This is an interface that can be used to retrieve information required to run the script (for example, information on trigger CIs and adapter parameters), and is also used to report on errors that occur during the script run. For details, see "[HP Data Flow Management API Reference](#)" on page 52.

The correct usage of Framework instance is to pass it as argument to each method that uses it.

Example:

```
def DiscoveryMain(Framework):
    OSHVResult = helperMethod (Framework)
    return OSHVResult
def helperMethod (Framework):
    ....
    probe_name      = Framework.getDestinationAttribute('probe_
name')
    ...
    return result
```

This section describes the most important Framework usages:

- "[Framework.getTriggerCIData\(String attributeName\)](#)" below
- "[Framework.createClient\(credentialsId, props\)](#)" on the next page
- "[Framework.getParameter \(String parameterName\)](#)" on page 61
- "[Framework.reportError\(String message\)](#) and [Framework.reportWarning\(String message\)](#)" on page 62

Framework.getTriggerCIData(String attributeName)

This API provides the intermediate step between the Trigger CI data defined in the adapter and the script.

Example of Retrieving Credential Information:

You request the following Trigger CI data information:

Triggered CI data	
Name	Value
Protocol	\${SOURCE.credentials_id}
credentialsId	\${SOURCE.credentials_id}
fileName	\${CONFIGURATION_DOCUMENT.name}
hostID	\${HOST.root_id}
ip_address	\${SOURCE.ip_address}
path	\${CONFIGURATION_DOCUMENT.resource_path}

To retrieve the credential information from the task, use this API:

```
credId = Framework.getTriggerCIData('credentialsId')
```

Framework.createClient(credentialsId, props)

You make a connection to a remote machine by creating a client object and executing commands on that client. To create a client, retrieve the `ClientFactory` class. The `getClientFactory()` method receives the type of the requested client protocol. The protocol constants are defined in the `ClientsConsts` class in the API documentation. For details on credentials and supported protocols, see Domain Credential References in the *Data Flow Management Guide*.

Example of Creating a Client Instance for the Credentials ID:

To create a `Client` instance for the credentials ID:

```
properties = Properties()
codePage = Framework.getCodePage()
properties.put( BaseAgent.ENCODING, codePage)
client = Framework.createClient(credentialsID ,properties)
```

You can now use the `Client` instance to connect to the relevant machine or application.

Example of Creating a WMI Client and Running a WMI Query:

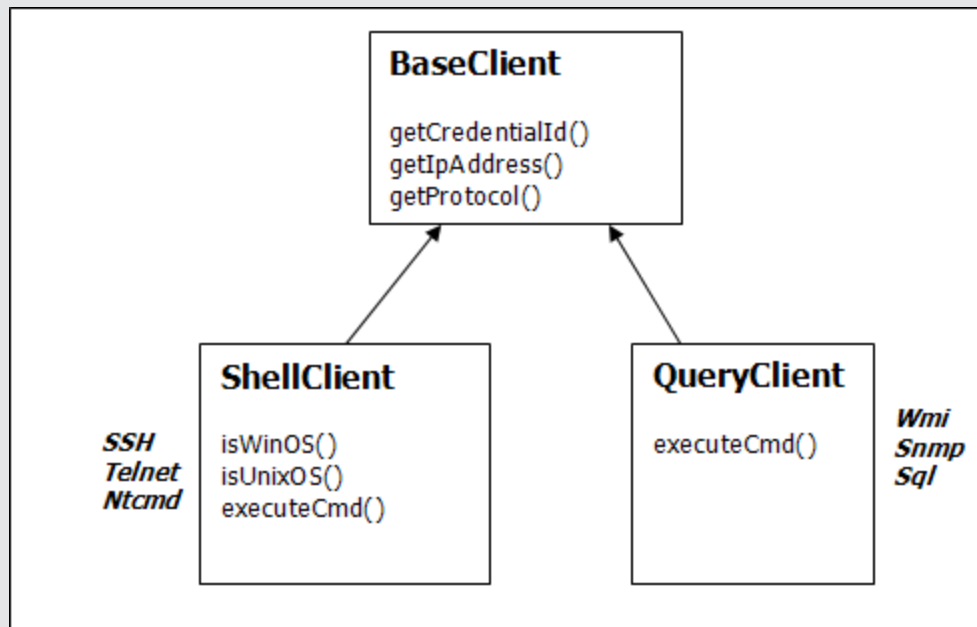
To create a WMI client and run a WMI query using the client:

```
wmiClient = Framework.createClient(credential)
resultSet = wmiClient.executeQuery("SELECT TotalPhysicalMemory

FROM Win32_LogicalMemoryConfiguration")
```

Note: To make the `createClient()` API work, add the following parameter to the Trigger CI data parameters: **credentialsId = \${SOURCE.credentials_id}** in the Triggered CI Data pane. Or you can manually add the credentials ID when calling the function:
wmiClient = clientFactory().createClient(credentials_id).

The following diagram illustrates the hierarchy of the clients, with their commonly-supported APIs:



For details on the clients and their supported APIs, see BaseClient, ShellClient, and QueryClient in the *HP Discovery and Dependency Mapping Schema Reference*. These files are located in the following folder:

\\<BSM root directory>\AppServer\webapps\site.war\amdocs\eng\API_docs\DDM_Schema\webframe.html

Framework.getParameter (String parameterName)

In addition to retrieving information on the Trigger CI, you often need to retrieve an adapter parameter value. For example:

Parameters		
Override	Name	Value
<input checked="" type="checkbox"/>	protocolType	MicrosoftSQLServer

Example of Retrieving the Value of the protocolType Parameter:

To retrieve the value of the `protocolType` parameter from the Jython script, use the following API:

```
protocolType = Framework.getParameterValue('protocolType')
```

Framework.reportError(String message) and Framework.reportWarning(String message)

Some errors (for example, connection failure, hardware problems, timeouts) can occur during a script run. When such errors are detected, Framework can report on the problem. The message that is reported reaches the server and is displayed for the user.

Example of a Report Error and Message:

The following example illustrates the use of the `reportError(<Error Msg>)` API:

```
try:
    client = Framework.getClientFactory(ClientsConsts.SNMP_
    PROTOCOL_NAME)
    createClient()
except:
    strException = str(sys.exc_info()[1]).strip()
    Framework.reportError('Connection failed: %s' %
    strException)
```

You can use either one of the APIs—`Framework.reportError(String message)`, `Framework.reportWarning(String message)`—to report on a problem. The difference between the two APIs is that when reporting an error, the Probe saves a communication log file with the entire session's parameters to the file system. In this way you are able to track the session and better understand the error.

Finding the Correct Credentials (for Connection Adapters)

An adapter trying to connect to a remote system needs to try all possible credentials. One of the parameters needed when creating a client (through `ClientFactory`) is the credentials ID. The connection script gains access to possible credential sets and tries them one by one using the `clientFactory.getAvailableProtocols()` method. When one credential set succeeds, the adapter reports a CI connection object on the host of this trigger CI (with the credentials ID that matches the IP) to the RTSM. Subsequent adapters can use this connection object CI directly to connect to the credential set (that is, the adapters do not have to try all possible credentials again).

The following example shows how to obtain all entries of the SNMP protocol. Note that here the IP is obtained from the Trigger CI data (# Get the Trigger CI data values).

The connection script requests all possible protocol credentials (# Go over all the protocol credentials) and tries them in a loop until one succeeds (`resultVector`). For details, see the **two-phase connect paradigm** entry in ["Separating Adapters" on page 27](#).

```
import logger
from appilog.collectors.clients import ClientsConsts
from appilog.common.system.types.vectors import
ObjectStateHolderVector
    def mainFunction(Framework):
resultVector = ObjectStateHolderVector()
    # Get the Trigger CI data values
    ip_address = Framework.getDestinationAttribute('ip_address')
```

```
ip_domain = Framework.getDestinationAttribute('ip_domain')
# Create the client factory for SNMP
clientFactory = framework.getClientFactory(ClientsConsts.SNMP_
PROTOCOL_NAME)
protocols = clientFactory.getAvailableProtocols(ip_address,
ip_domain)

connected = 0
# Go over all the protocol credentials
for credentials_id in protocols:
    client = None
    try:
        # try to connect to the snmp agent
        client = clientFactory.createClient(credentials_id)
        // Query the agent
        ....
        # connection succeed
        connected = 1
    except:
        if client != None:
            client.close()
if (not connected):
    logger.debug('Failed to connect using all credentials')
else:
    // return the results as OSHV
    return resultVector
```

Handling Exceptions from Java

Some Java classes throw an exception upon failure. It is recommended to catch the exception and handle it, otherwise it causes the adapter to terminate unexpectedly.

When catching a known exception, in most cases you should print its stack trace to the log and issue a proper message to the UI, for example:

```
try:
    client = Framework.getClientFactory().createClient()
except Exception, msg:
    Framework.reportError('Connection failed')
    logger.debugException('Exception while connecting: %s' %
(msg))
    return
```

If the exception is not fatal and the script can continue, you should omit the call for the `reportError()` method and enable the script to continue.

Support Localization in Jython Adapters

The multi-lingual locale feature enables DFM to work across different operating system (OS) languages, and to enable appropriate customizations at runtime.

Previously, before Content Pack 3.00, DFM used statically-specified encoding to treat output from all network targets. However, this approach does not suit a multi-lingual IT network: to discover hosts with different OS languages, Probe administrators had to re-run DFM jobs manually several times with different job parameters each time. This procedure produced a serious overhead on network load but, even more, it avoided several key features of DFM, such as immediate job invocation on a trigger CI or automatic data refreshing in RTSM by the Schedule Manager.

The following locale languages are supported by default: Japanese, Russian, and German. The default locale is English.

This section includes:

- ["Add Support for a New Language" below](#)
- ["Change the Default Language" on the next page](#)
- ["Determine the Character Set for Encoding" on page 66](#)
- ["Define a New Job to Operate With Localized Data" on page 66](#)
- ["Decode Commands Without a Keyword" on page 67](#)
- ["Work with Resource Bundles" on page 68](#)
- ["API Reference" on page 69](#)

Add Support for a New Language

This task describes how to add support for a new language.

This task includes the following steps:

- ["Add a Resource Bundle \(*.properties Files\)" below](#)
- ["Declare and Register the Language Object" on the next page](#)

1. Add a Resource Bundle (*.properties Files)

Add a resource bundle according to the job that is to be run. The following table lists the DFM jobs and the resource bundle that is used by each job:

Job	Base Name of Resource Bundle
File Monitor by Shell	langFileMonitoring
Host Resources and Applications by Shell	langHost_Resources_By_TTY, langTCP

Job	Base Name of Resource Bundle
Hosts by Shell using NSLOOKUP in DNS Server	langNetwork
Host Connection by Shell	langNetwork
Collect Network Data by Shell or SNMP	langTCP
Host Resources and Applications by SNMP	langTCP
Microsoft Exchange Connection by NTCMD, Microsoft Exchange Topology by NTCMD	msExchange
MS Cluster by NTCMD	langMsCluster

For details on bundles, see ["Work with Resource Bundles" on page 68](#).

2. Declare and Register the Language Object

To define a new language, add the following two lines of code to the **shellutils.py** script, that currently contains the list of all supported languages. The script is included in the `AutoDiscoveryContent` package. To view the script, access the Adapter Management window. For details, see Adapter Management Window in the Data Flow Management Guide.

- a. Declare the language, as follows:

```
LANG_RUSSIAN = Language(LOCALE_RUSSIAN, 'rus', ('Cp866',  
'Cp1251'), (1049,), 866)
```

For details on class language, see ["API Reference" on page 69](#). For details on the Class Locale object, see <http://java.sun.com/j2se/1.5.0/docs/api/java/util/Locale.html>. You can use an existing locale or define a new locale.

- b. Register the language by adding it to the following collection:

```
LANGUAGES = (LANG_ENGLISH, LANG_GERMAN, LANG_SPANISH, LANG_  
RUSSIAN, LANG_JAPANESE)
```

Change the Default Language

If the OS language cannot be determined, the default one is used. The default language is specified in the **shellutils.py** file.

```
#default language for fallback  
DEFAULT_LANGUAGE = LANG_ENGLISH
```

To change the default language, you initialize the `DEFAULT_LANGUAGE` variable with a different language. For details, see ["Add Support for a New Language" on the previous page](#).

Determine the Character Set for Encoding

The suitable character set for decoding command output is determined at runtime. The multi-lingual solution is based on the following facts and assumptions:

1. It is possible to determine the OS language in a locale-independent way, for example, by running the **chcp** command on Windows or the **locale** command on Linux.
2. Relation Language-Encoding is well known and can be defined statically. For example, the Russian language has two of the most popular encoding: Cp866 and Windows-1251.
3. One character set for each language is preferable, for example, the preferable character set for Russian language is Cp866. This means that most of the commands produce output in this encoding.
4. Encoding in which the next command output is provided is unpredictable, but it is one of the possible encoding for a given language. For example, when working with a Windows machine with a Russian locale, the system provides the **ver** command output in Cp866, but the **ipconfig** command is provided in Windows-1251.
5. A known command produces known key words in its output. For example, the **ipconfig** command contains the translated form of the **IP-Address** string. So the **ipconfig** command output contains **IP-Address** for the English OS, **IP-Adpec** for the Russian OS, **IP-Adresse** for the German OS, and so on.

Once it is discovered in which language the command output is produced (point 1 above), possible character sets are limited to one or two (point 2 above). Furthermore, it is known which key words are contained in this output (point 5 above).

The solution, therefore, is to decode the command output with one of the possible encoding by searching for a key word in the result. If the key word is found, the current character set is considered the correct one.

Define a New Job to Operate With Localized Data

This task describes how to write a new job that can operate with localized data.

Jython scripts usually execute commands and parse their output. To receive this command output in a properly decoded manner, use the API for the **ShellUtils** class. For details, see "[RTSM \(HP Universal CMDB\) Web Service API Overview](#)" on page 260.

This code usually takes the following form:

```
client = Framework.createClient(protocol, properties)
shellUtils = shellutils.ShellUtils(client)
languageBundle = shellutils.getLanguageBundle ('langNetwork',
shellUtils.osLanguage, Framework)
strWindowsIPAddress = languageBundle.getString('windows_ipconfig_
str_ip_address')
ipconfigOutput = shellUtils.executeCommandAndDecode('ipconfig
/all', strWindowsIPAddress)
```

```
#Do work with output here
```

1. Create a client:

```
client = Framework.createClient(protocol, properties)
```

2. Create an instance of the **ShellUtils** class and add the operating system language to it. If the language is not added, the default language is used (usually English):

```
shellUtils = shellutils.ShellUtils(client)
```

During object initialization, DFM automatically detects machine language and sets preferable encoding from the predefined `Language` object. Preferable encoding is the first instance appearing in the encoding list.

3. Retrieve the appropriate resource bundle from **shellClient** using the **getLanguageBundle** method:

```
languageBundle = shellutils.getLanguageBundle ('langNetwork',  
shellUtils.osLanguage, Framework)
```

4. Retrieve a keyword from the resource bundle, suitable for a particular command:

```
strWindowsIPAddress = languageBundle.getString('windows_ipconfig_  
str_ip_address')
```

5. Invoke the **executeCommandAndDecode** method and pass the keyword to it on the **ShellUtils** object:

```
ipconfigOutput = shellUtils.executeCommandAndDecode('ipconfig  
/all', strWindowsIPAddress)
```

The `ShellUtils` object is also needed to link a user to the API reference (where this method is described in detail).

6. Parse the output as usual.

Decode Commands Without a Keyword

The current approach for localization uses a keyword to decode all of the command output. For details, see the step ["Retrieve a keyword from the resource bundle, suitable for a particular command:"](#) in ["Define a New Job to Operate With Localized Data"](#) on the previous page.

However, another approach uses a keyword to decode the first command output only, and then decodes further commands with the character set used to decode the first command. To do this, you use the **getCharsetName** and **useCharset** methods of the **ShellUtils** object.

The regular use case works as follows:

1. Invoke the **executeCommandAndDecode** method once.
2. Obtain the most recently used character set name through the **getCharsetName** method.
3. Make **shellUtils** use this character set by default, by invoking the **useCharset** method on the **ShellUtils** object.

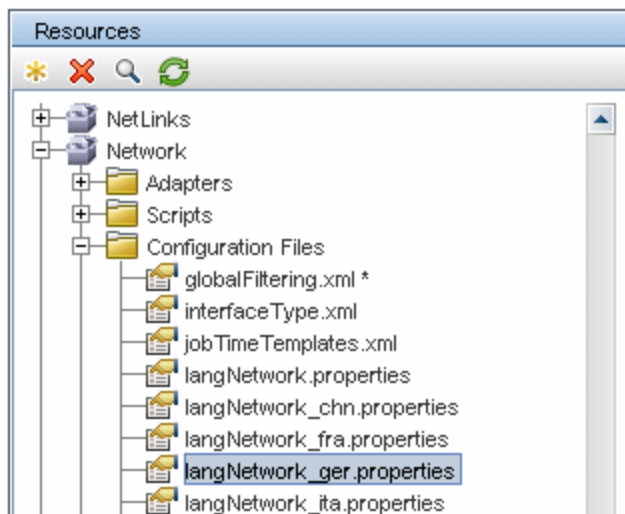
4. Invoke the **execCmd** method of **ShellUtils** one or more times. The output is returned with the character set specified in the previous step. No additional decoding operations occur.

Work with Resource Bundles

A resource bundle is a file that takes a properties extension (*.properties). A properties file can be considered a dictionary that stores data in the format of `key = value`. Each row in a properties file contains one `key = value` association. The main functionality of a resource bundle is to return a value by its key.

Resource bundles are located on the Probe machine:

C:\hp\UCMDB\DataFlowProbe\runtime\probeManager\discoveryConfigFiles. They are downloaded from the RTSM Server as any other configuration file. They can be edited, added, or removed, in the Resources window. For details, see Configuration File Pane in the Data Flow Management Guide.



When discovering a destination, DFM usually needs to parse text from command output or file content. This parsing is often based on a regular expression. Different languages require different regular expressions to be used for parsing. For code to be written once for all languages, all language-specific data must be extracted to resource bundles. There is a resource bundle for each language. (Although it is possible that a resource bundle contain data for different languages, in DFM one resource bundle always contains data for one language.)

The Jython script itself does not include hard coded, language-specific data (for example, language-specific regular expressions). The script determines the language of the remote system, loads the proper resource bundle, and obtains all language-specific data by a specific key.

In DFM, resource bundles take a specific name format: `<base_name>_<language_identifier>.properties`, for example, `langNetwork_spa.properties`. (The default resource bundle takes the following format: `<base_name>.properties`, for example, `langNetwork.properties`.)

The `base_name` format reflects the intended purpose of this bundle. For example, **langMsCluster** means the resource bundle contains language-specific resources used by the MS Cluster jobs.

The `language_identifier` format is a 3-letter acronym used to identify the language. For example, `rus` stands for the Russian language and `ger` for the German language. This language identifier is included in the declaration of the `Language` object.

API Reference

This section includes:

- ["The Language Class" below](#)
- ["The executeCommandAndDecode Method" below](#)
- ["The getCharsetName Method" on the next page](#)
- ["The useCharset Method" on the next page](#)
- ["The getLanguageBundle Method" on the next page](#)
- ["The osLanguage Field" on the next page](#)

The Language Class

This class encapsulates information about the language, such as resource bundle postfix, possible encoding, and so on.

Fields

Name	Description
<code>locale</code>	Java object which represents locale.
<code>bundlePostfix</code>	Resource bundle postfix. This postfix is used in resource bundle file names to identify the language. For example, the langNetwork_ger.properties bundle includes a ger bundle postfix.
<code>charsets</code>	Character sets used to encode this language. Each language can have several character sets. For example, the Russian language is commonly encoded with the <code>Cp866</code> and <code>Windows-1251</code> encoding.
<code>wmiCodes</code>	The list of WMI codes used by the Microsoft Windows OS to identify the language. All possible codes are listed at http://msdn.microsoft.com/en-us/library/aa394239(VS.85).aspx (the <code>OSLanguage</code> section). One of the methods for identifying the OS language is to query the WMI class <code>OS</code> for the <code>OSLanguage</code> property.
<code>codepage</code>	Code page used with a specific language. For example, <code>866</code> is used for Russian machines and <code>437</code> for English machines. One of the methods for identifying the OS language is to retrieve its default codepage (for example, by the <code>chcp</code> command).

The executeCommandAndDecode Method

This method is intended to be used by business logic Jython scripts. It encapsulates the decoding operation and returns a decoded command output.

Arguments

Name	Description
cmd	The actual command to be executed.
keyword	The keyword to be used for the decoding operation.
framework	The Framework object passed to every executable Jython script in DFM.
timeout	The command timeout.
waitForTimeout	Specifies if client should wait when timeout is exceeded.
useSudo	Specifies if <code>sudo</code> should be used (relevant only for UNIX machine clients).
language	Enables specifying the language directly instead of automatically detecting a language.

The `getCharsetName` Method

This method returns the name of the most recently used character set.

The `useCharset` Method

This method sets the character set on the `ShellUtils` instance, which uses this character set for initial data decoding.

Name	Description
charsetName	The name of the character set, for example, <code>windows-1251</code> or <code>UTF-8</code> .

See also "[The `getCharsetName` Method](#)" above.

The `getLanguageBundle` Method

This method should be used to obtain the correct resource bundle. This replaces the following API:

```
Framework.getEnvironmentInformation().getBundle(...)
```

Name	Description
baseName	The name of the bundle without the language suffix, for example, <code>langNetwork</code> .
language	The language object. The <code>ShellUtils.osLanguage</code> should be passed here.
framework	The Framework, common object which is passed to every executable Jython script in DFM.

The `osLanguage` Field

This field contains an object that represents the language.

Work with Discovery Analyzer

The Discovery Analyzer tool is intended for debugging purposes when developing packages, scripts, or any other content. The tool runs a job against a remote destination and returns logs containing information, warning, and error details and results of discovered CIs.

Note that results are not always reported to the UI. This is because the results are reported in two ways and only one of them is supported. Also, the communication log is not supported from Eclipse.

When executing the tool from Eclipse, the **DiscoveryProbe.properties** file (**C:\hp\UCMDB\DataFlowProbe\conf\DiscoveryProbe.properties**) must contain the following parameter set to **true**:

```
appilog.agent.local.discoveryAnalyzerFromEclipse = true
```

For details, see ["Run Discovery Analyzer from Eclipse" on page 77](#).

In all other cases (when the tool is executed from the **cmd** file or while the Probe is running) this flag must be set to **false**:

```
appilog.agent.local.discoveryAnalyzerFromEclipse = false
```

Tasks and Records

A task file contains data regarding a task to be executed. The task consists of information such as the job's name and required parameters that define the trigger CI, for example, the remote destination address.

A record file contains task information as well as the results of a specific execution, that is, the detailed communication (including a response) between the Probe or Discovery Analyzer (whichever module executed the task) and the remote destination.

A task that is defined by a task file can be executed against a remote destination, whereas a task that is defined by a record file (that contains extra data regarding a specific execution) can be executed and can also be played back (that is, can reproduce the same execution documented in the record file).

Logs

Logs provide information about the latest run, as follows:

- **General Log.** This log includes all information data, errors, and warnings that occurred during the run.
- **Communication Log.** This log contains the detailed communication between the Discovery Analyzer and the remote destination (including its response). After the execution, the log can be saved as a record file.
- **Results Log.** Displays a list of discovered CIs. The appearance time of each CI depends on the design of the adapters and scripts.

You can save all logs together or each log separately. When you save all the logs, they are saved together under one name.

If you replay a record file, the same data is displayed in the communication log, the only difference being the time of execution.

Limitation: The Communication and Results logs are not available when running Discovery Analyzer through Eclipse.

This section includes the following steps:

- ["Prerequisites" below](#)
- ["Access Discovery Analyzer" below](#)
- ["Define a Task" on the next page](#)
- ["Define a New Task" on the next page](#)
- ["Retrieve a Record" on page 74](#)
- ["Open a Task File" on page 75](#)
- ["Import a Task from the Database" on page 75](#)
- ["Edit a Task" on page 75](#)
- ["Save the Task and Logs" on page 75](#)
- ["Run the Task" on page 75](#)
- ["Send a Task Result to the Server" on page 75](#)
- ["Import Settings" on page 76](#)
- ["Breakpoints" on page 76](#)
- ["Configure Eclipse" on page 76](#)

1. **Prerequisites**

- The Probe must be installed. (The Discovery Analyzer is installed as part of the Probe installation process and shares resources with it.)
- The Probe does not need to be running while you are working with Discovery Analyzer.

However, if the Probe has already run against an RTSM Server, all the required resources are already downloaded to the file system. If the Probe has not run, you can upload resources needed by Discovery Analyzer through the Settings menu. For details, see ["Import Settings" on page 76](#).

- The RTSM Server does not need to be installed.

2. **Access Discovery Analyzer**

You access Discovery Analyzer either:

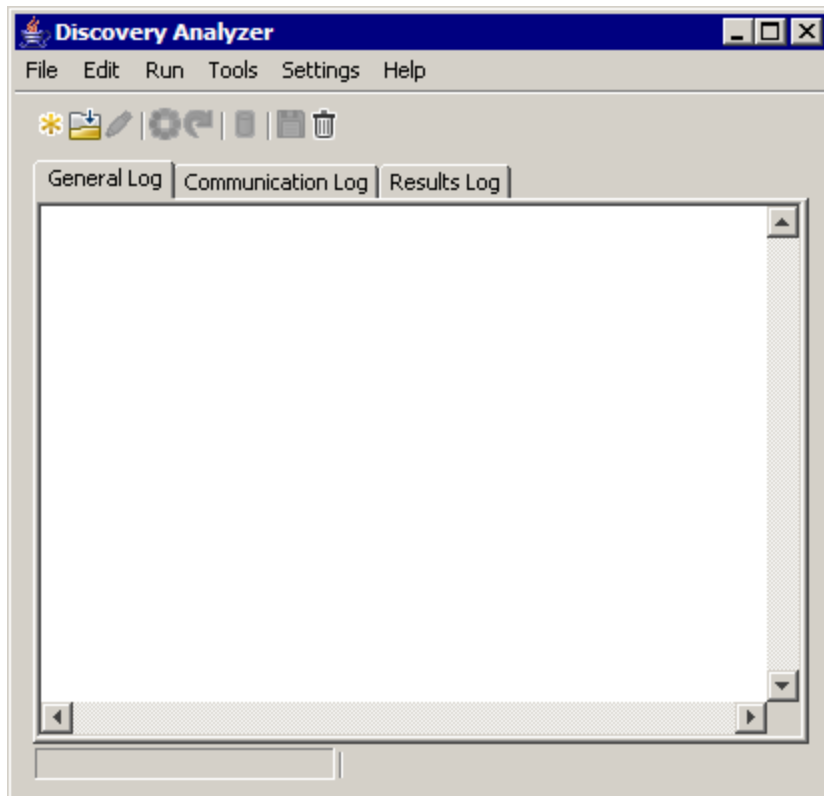
- When working with Eclipse.

The Probe installation comes with a default Eclipse workspace located at **C:\hp\UCMDB\DataFlowProbe\tools\discoveryAnalyzerWorkspace**. This workspace

includes a Jython script to start Discovery Analyzer (**startDiscoveryAnalyzerScript.py**) as well as a link to all DFM scripts. If you start the tool in this way, you can locate breakpoints within the Jython scripts for debugging purposes.

- Directly, by double-clicking the file in the following folder:
C:\hp\UCMDB\DataFlowProbe\tools\discoveryAnalyzer.cmd. For details, see ["Run Discovery Analyzer from Eclipse" on page 77](#).

The Discovery Analyzer window opens:




3. Define a Task

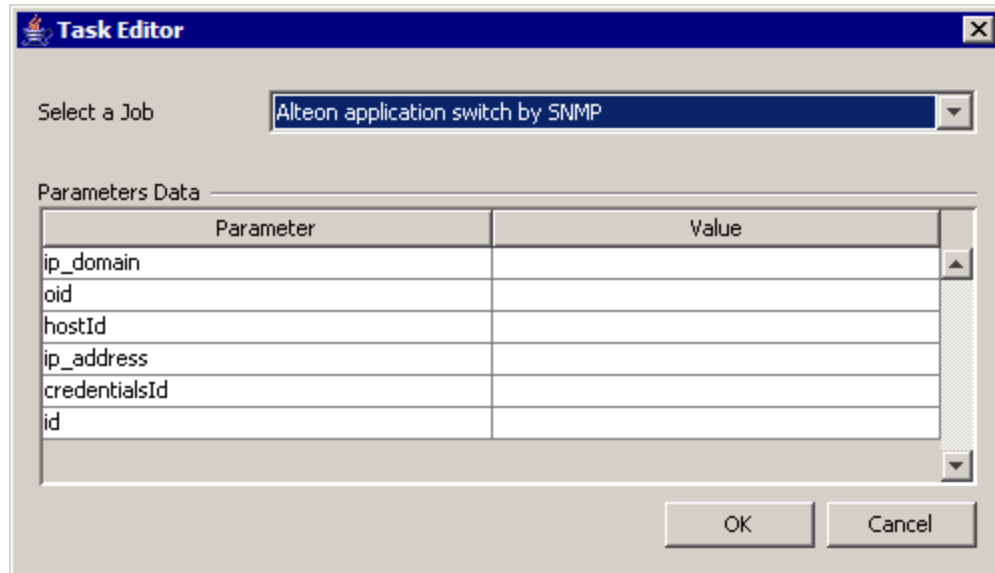
You define a task using one of the following methods:

- By defining a new task. For details, see ["Define a New Task" below](#).
- By importing a task from a record file. For details, see ["Retrieve a Record" on the next page](#).
- By importing a saved task from a task file. For details, see ["Open a Task File" on page 75](#).
- By retrieving a job from the Probe's internal database. For details, see ["Import a Task from the Database" on page 75](#).

4. Define a New Task

- a. Display the Task Editor: click the **New Task** button .

The Task Editor displays a list of jobs that currently exist in the file system. This list is updated each time the Probe receives tasks from the server, or packages are deployed manually from the Settings menu.



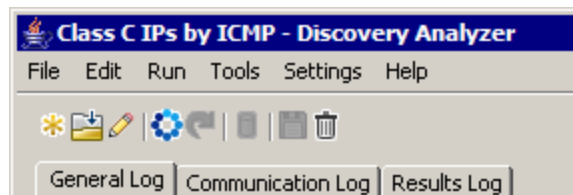
- b. Select a job.
- c. Enter values for all parameters.

The parameters displayed here are DFM adapter parameters. They can be viewed in the Discovery Pattern Parameters pane in the Pattern Signature tab. For details, see Adapter Definition Tab in the *Data Flow Management Guide*.

All fields are mandatory (unless a job's script demands that the field be empty).

For parameters that require an ID or credentials ID input value, you can use randomly created IDs: right-click the value box and select **Generate random CMDB ID** or **Credential Chooser**.

The task is now active and the name of the open task is displayed in the title bar:



- d. Continue with the procedure for defining a task. For details, see ["Save the Task and Logs" on the next page](#).

5. Retrieve a Record

You can define a task by opening a record file containing data regarding a specific execution. If a task is defined in this way, you can reproduce the specific execution by selecting the playback option. (If a task is replayed, responses are received from the data stored in the record file and not from the remote destination.)

Select **File > Open Record**. Browse to the folder where you saved the record. The record is now active and the name of the task is displayed in the title bar.

For details on acquiring a record file, see ["Record DFM Code" on page 87](#).

6. Open a Task File

You can define a task from a task file: Select **File > Open Task**.

7. Import a Task from the Database

You can retrieve a task from the Probe database on condition that the Probe has already run and has active tasks in its internal database. You can use the parameter values to define the task.

- a. Select **File > Import Task from Probe Database**.
- b. In the dialog box that opens, select the task to run and click **OK**.
- c. Continue with the procedure for defining a task. For details, see ["Save the Task and Logs" below](#).

8. Edit a Task

After a task is defined, the name of the task (or the file) is displayed in the title bar. Now the file can be edited.

- a. Select **Edit > Edit Task**.
- b. Make any changes to the task and click **OK**.

9. Save the Task and Logs

You can save task parameters: Select **File > Save Task**.

The following options are available only after a task is executed:

- Save a record of the task. You can save the task parameters and the results of the task run: Select **File > Save Record**.
- Save a log of the task: Select **File > Save General Log**.
- Save results: Select **File > Save Results**.

10. Run the Task

The next step in the procedure is to run the task you created.

- a. Import the credentials/ranges configuration file. For details, see ["Import Settings" on the next page](#).
- b. To execute the task only against a remote destination, click the **Run Task** button.

Discovery Analyzer executes the job and displays information in the three log files: **General**, **Communication**, and **Results**.
- c. You can save the log files, either together or separately: Select **File > Save General Log**, **Save Record**, **Save Results**, or **Save All Logs**. For details on the log files, see ["Logs" on page 71](#).
- d. If a task is retrieved from a record file, the execution that is documented in this file can be reproduced by clicking the **Playback** button. The same Communication log is displayed, but the execution time is updated.

11. Send a Task Result to the Server

If a task's execution ends with results (that is, the Results Log tab displays a list of discovered CIs), you can send the results to the RTSM Server. This is useful if, for example, you were previously testing a script when the server was down.

Note: You can send results only to an RTSM Server that receives tasks from the Probe that is installed on the same machine as Discovery Analyzer.

12. Import Settings

To run tasks or the playback record file, you must import the **domainScopeDocument.bin** file. During import, you enter a password.

- a. Launch a Web browser and enter the following URL: **http://localhost:8080/jmx-console**. You may have to log in with a user name and password.
- b. Click **UCMDB:service=DiscoveryManager** to open the JMX MBEAN View page.
- c. Locate the **exportCredentialsAndRangesInformation** operation. Do the following:
 - o Enter the customer ID (the default is **1**).
 - o Enter a name for the exported file.
 - o Enter the password.
 - o Set **isEncrypted** to **False**.
- d. Click **Invoke** to export the **domainScopeDocument.bin** file.

When the export process completes successfully, the file is saved to the following location:
C:\hp\UCMDB\UCMDBServer\conf\discovery\<customer_dir>.

- e. Copy the **domainScopeDocument.bin** file to the Data Flow Probe file system and import it by selecting: **Settings > Import domainScopeDocument**.

Note: During the **domainScopeDocument** file import, you are requested to provide a password. This request is also displayed following each Discovery Analyzer restart and before the first task or record is executed.

13. Breakpoints

If you run Discovery Analyzer from the Python script, you can add breakpoints to your script.

14. Configure Eclipse

For details on running your Jython scripts in debug mode, see ["Run Discovery Analyzer from Eclipse" on the next page](#).

Run Discovery Analyzer from Eclipse

This task explains how to configure Eclipse so that you can run your Jython scripts in debug mode, thus enabling better visibility to job threads, trigger CIs, and results.

This section includes the following steps:

- "Prerequisites" below
- "Unpack Eclipse and start it." below
- "Configure the default workspace" below
- "Configure the PyDev Extensions" below
- "Configure the Discovery Analyzer Workspace" on page 79
- "Configure the classpath and interpreter" on page 83
- "Run Discovery Analyzer" on page 85

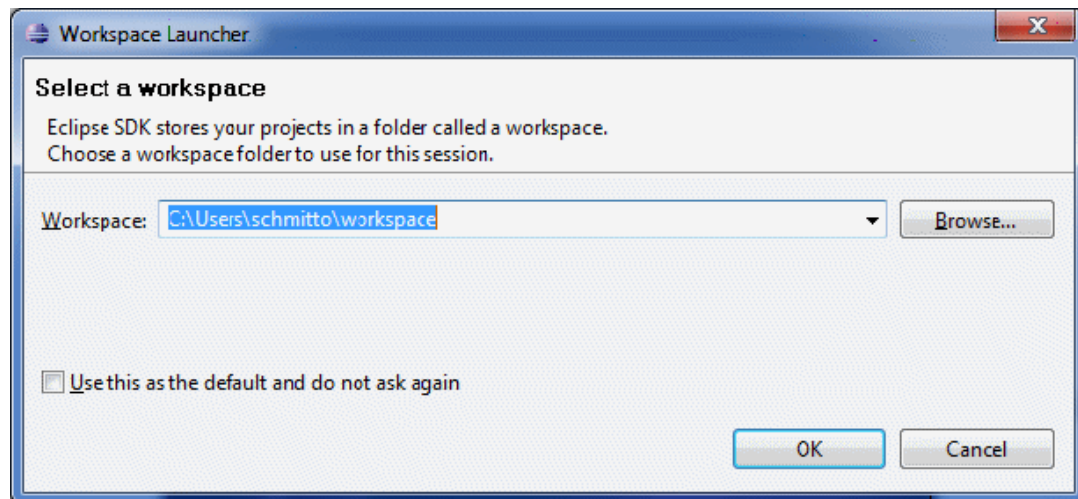
1. Prerequisites

- Install the latest Eclipse version on your computer. The application is available at www.eclipse.org.
- Verify that the Data Flow Probe is installed on the same computer.
- Verify that the **appilog.agent.local.discoveryAnalyzerFromEclipse** parameter in the **DiscoveryProbe.properties** file is set to **true**.

2. Unpack Eclipse and start it.

3. Configure the default workspace

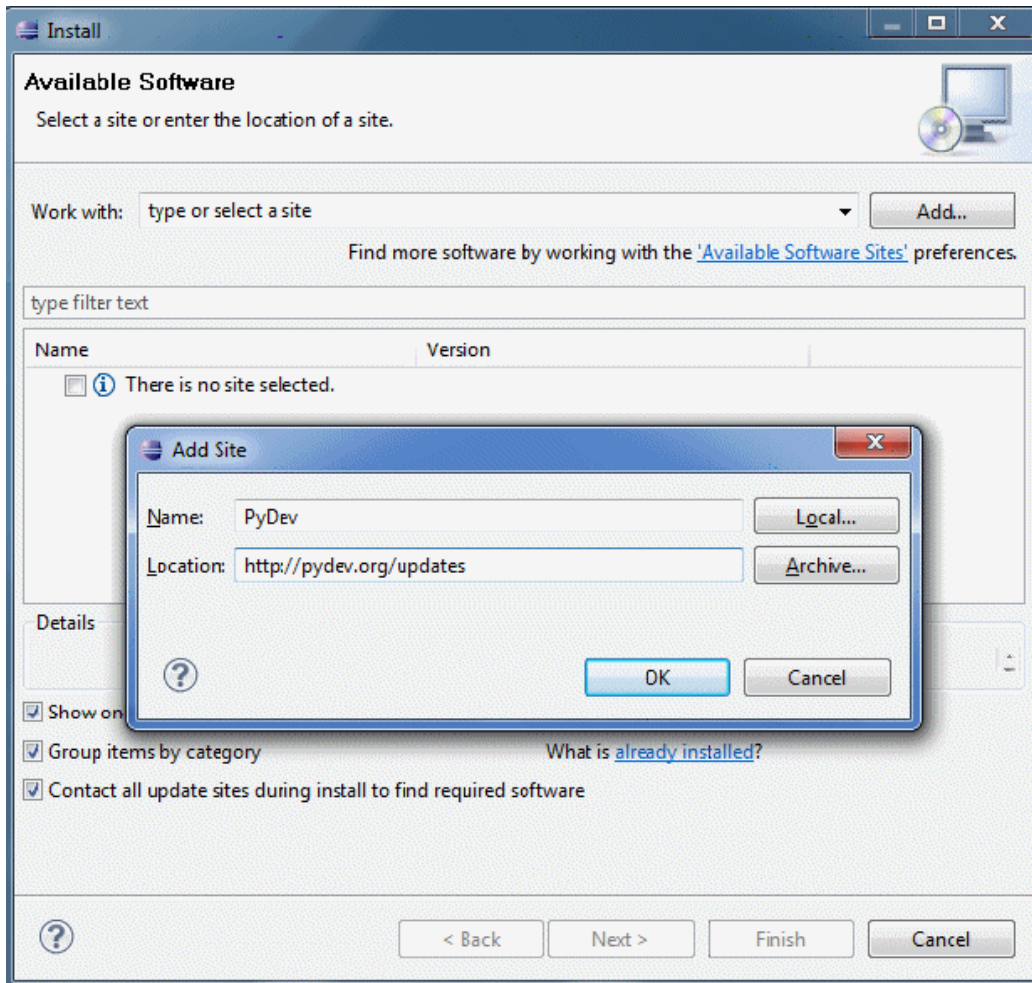
Configure the default workspace where Eclipse saves and stores all projects and the related data.



4. Configure the PyDev Extensions

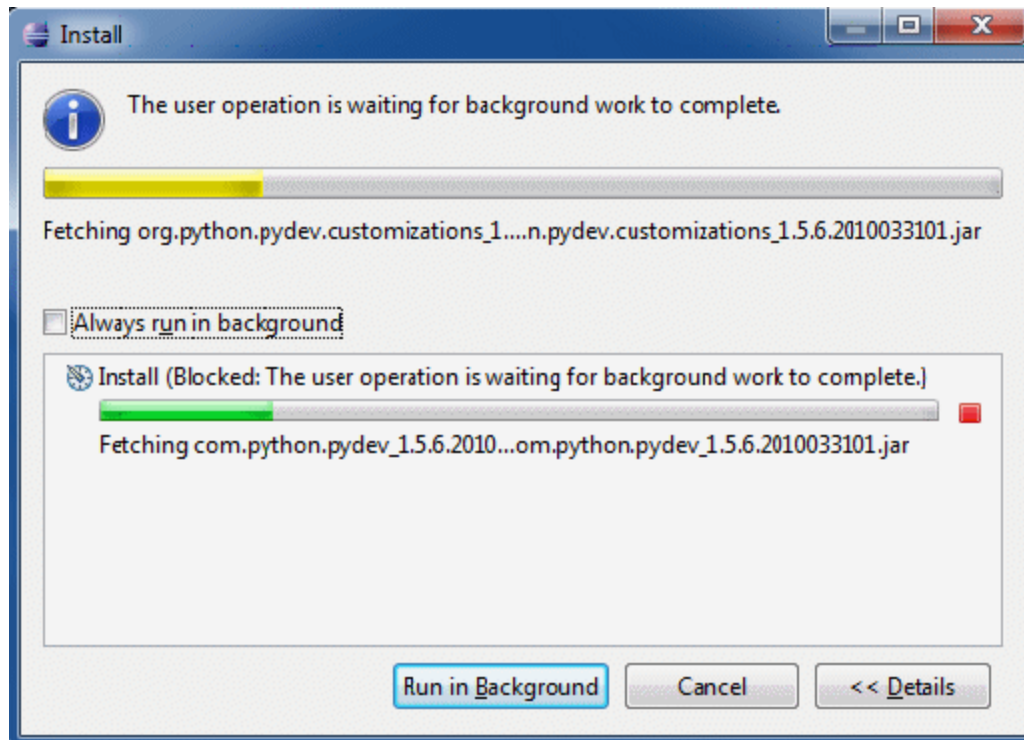
- a. Access **Help > Install New Software**, click **Add**, type a name for the PyDev plugin, and

in the Location field add the URL of the site where **pydev** can be downloaded:
<http://pydev.org/updates>. Click **OK**.



Note: PyDev and PyDev Extensions are now merged into one plugin since PyDev Extensions are now open source. For additional information visit <http://pydev.org>.

- b. In the window that opens, select **Pydev**. The second plugin is a plugin for task-focused UI. Click **Next**, check the installation details, and click **Next** again.
- c. Accept the license agreement and click **Next**.
- d. Pydev is installed. If you are asked to install unsigned content, confirm by clicking **OK**.

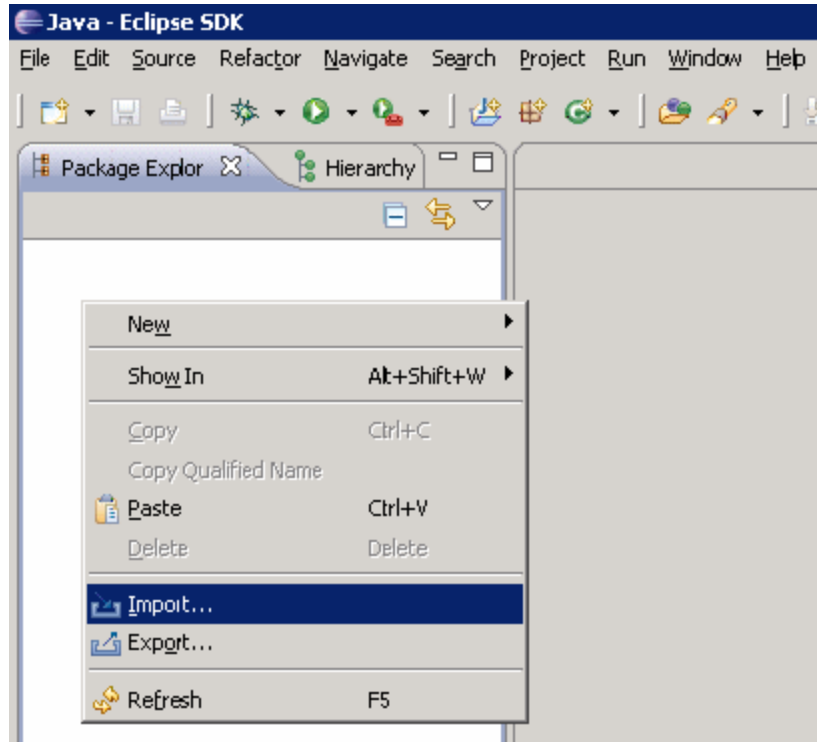


- e. Restart Eclipse.

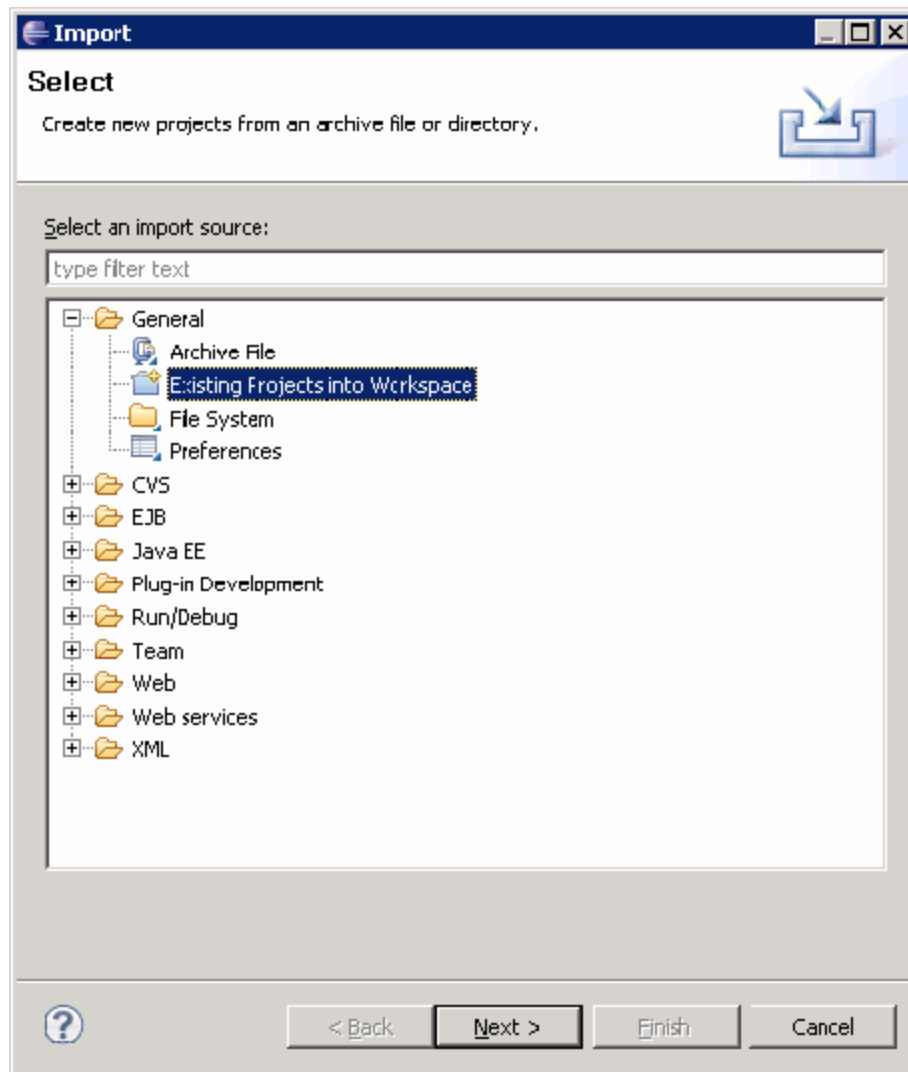
PyDev is now installed in your Eclipse IDE. You have new perspectives in Eclipse and the IDE is able to interpret Python scripts (text highlighting, additional configuration options, and so on).

5. Configure the Discovery Analyzer Workspace

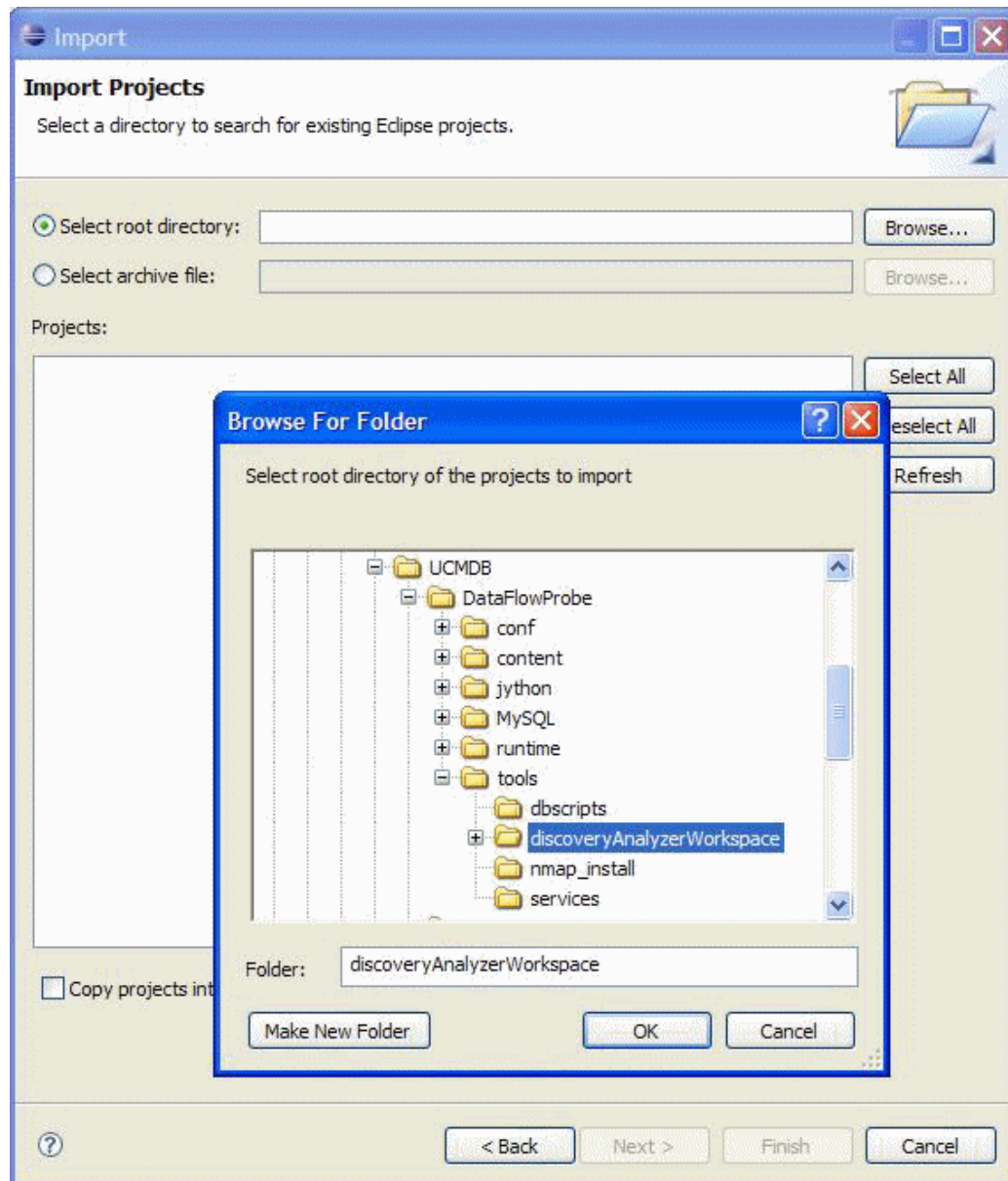
- a. Import necessary files: Right-click in the white area in Package Explorer and click **Import** to import the pre-configured **discoveryAnalyzerWorkspace**, included with the Probe installation.

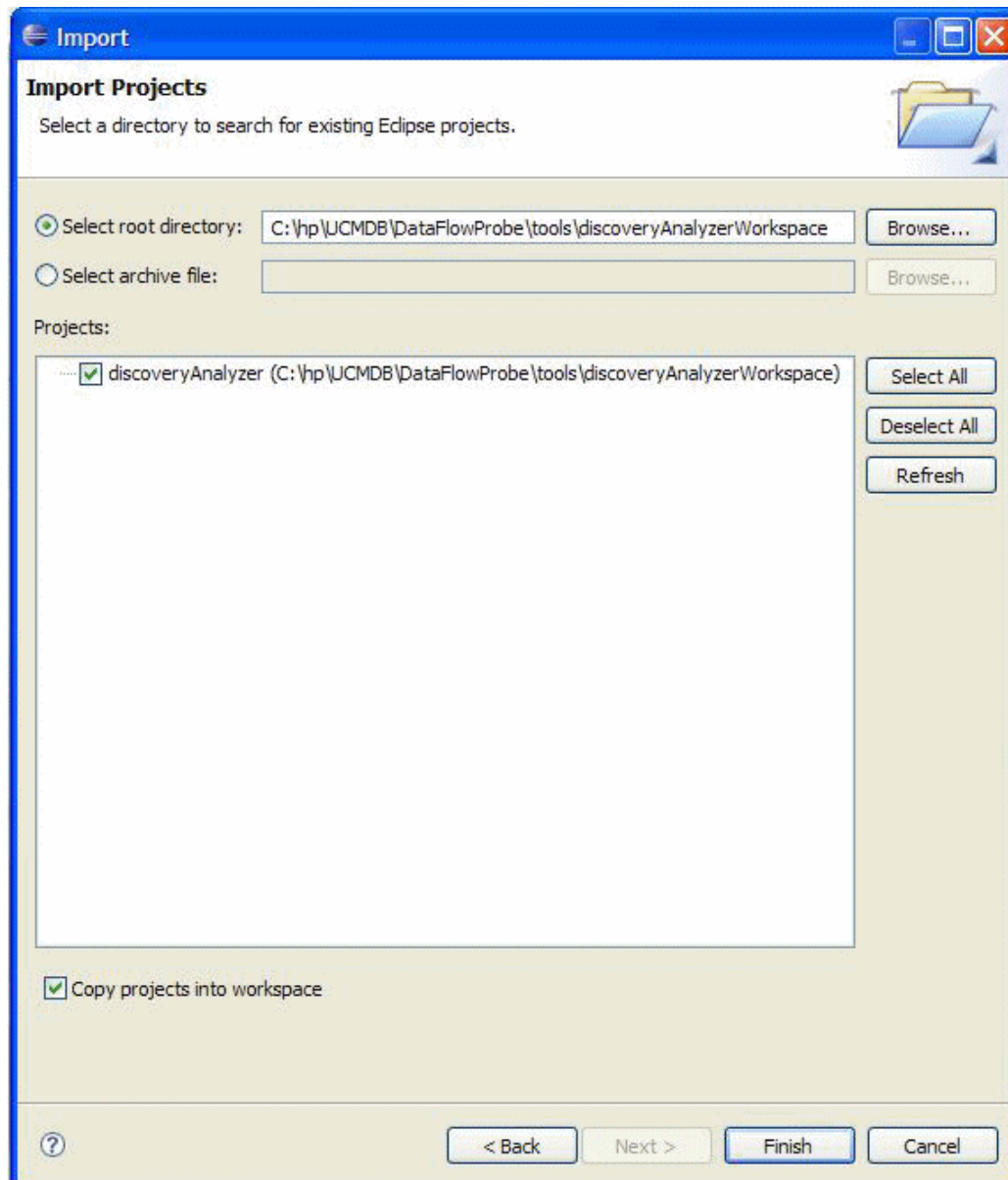


- b. Under **General**, select **Existing projects into Workspace** to import the project into the Eclipse workspace.



- c. Under **Select root directory**, select the Analyzer workspace, usually located under:
C:\hp\UCMDB\DataFlowProbe\tools\discoveryAnalyzerWorkspace.
- d. Select **Copy projects into workspace** to create a real copy of the existing workspace.
This is an important step: In case of failure, you can re-import the original **discoveryAnalyserWorkspace.**
- e. Click **Finish** to start the import.



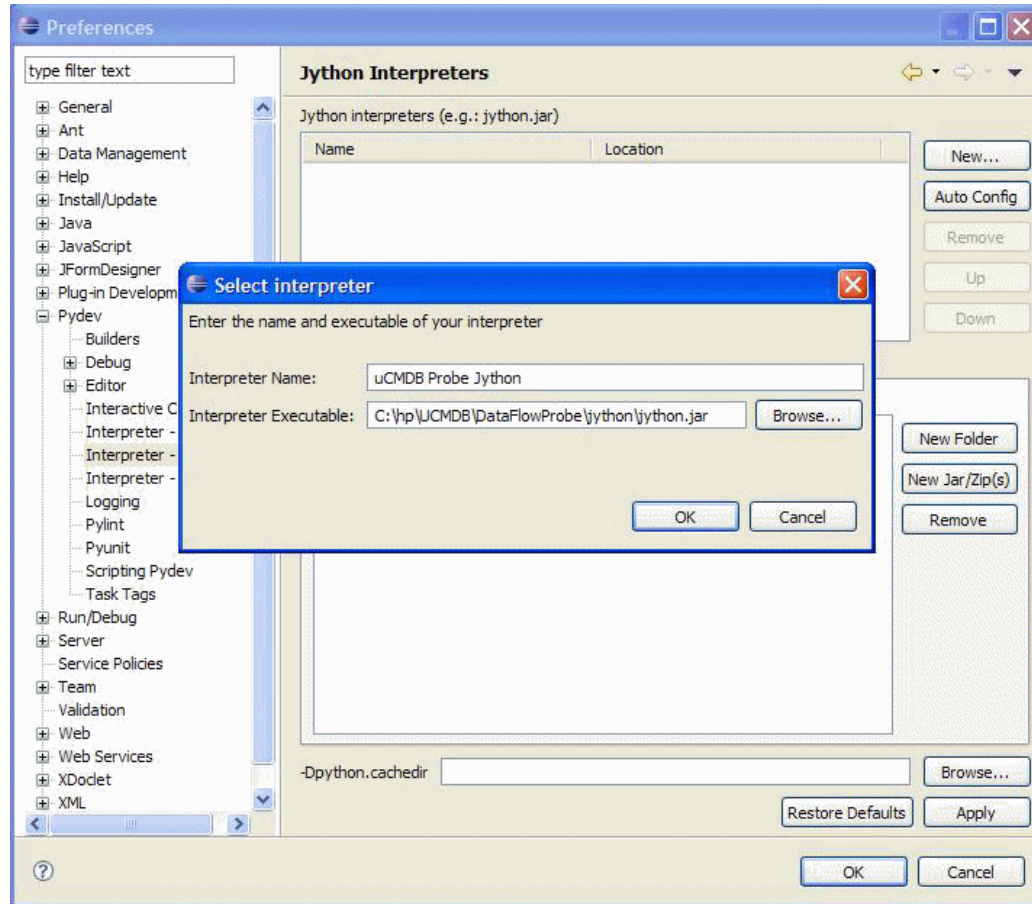


6. Configure the classpath and interpreter

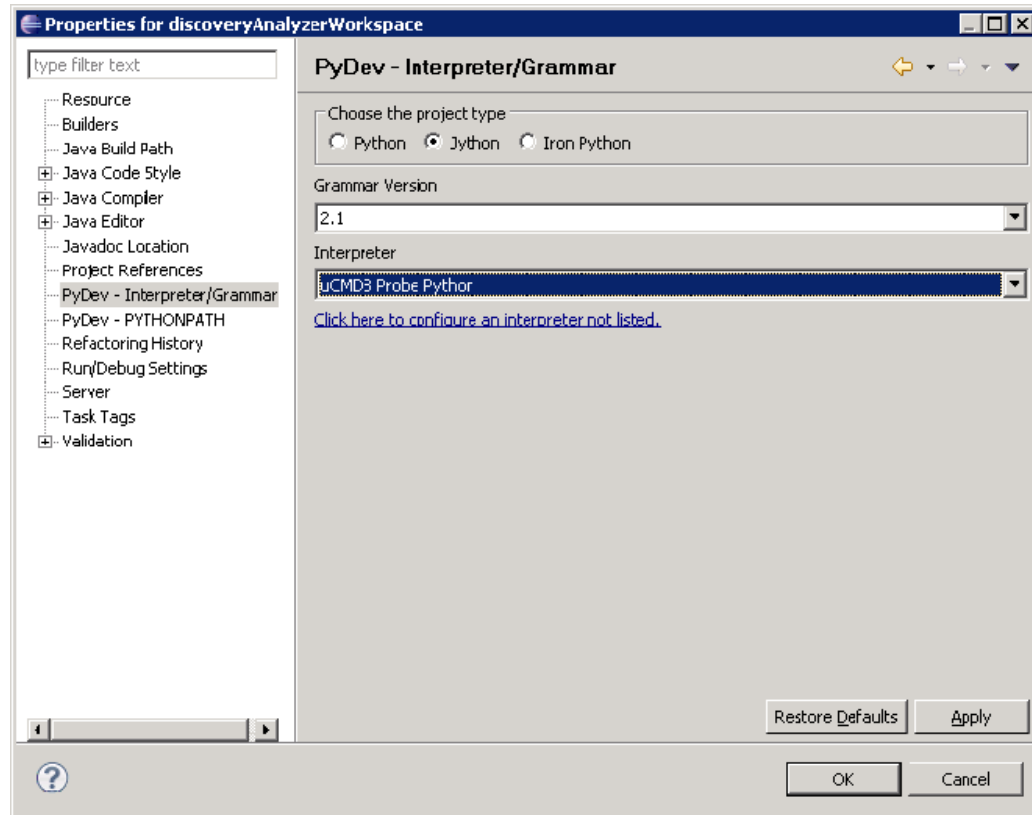
- Right-click **discoveryAnalyzerWorkspace** and select **Properties** to display the Project specific settings.
- Go to **Pydev > Interpreter/Grammar** and click **Please configure an interpreter in the related preferences before proceeding**.

This step configures the same Jython interpreter as the Probe is using to ensure that scripts are not interpreted by a different Jython version.

- Click **New**, type a name for the interpreter, and select the file from the following folder:
C:\hp\UCMDB\DataFlowProbe\jython\jython.jar.



- d. Click **OK**. If a window is displayed, asking you to select the folders that should be imported into your Python system path, do not change anything (should be **C:\hp\UCMDB\DataFlowProbe\jython** and **C:\hp\UCMDB\DataFlowProbe\jython\lib**) and click **OK**.
- e. Click **Apply** and then **OK**.
- f. Click **Interpreter** and select the interpreter just created.

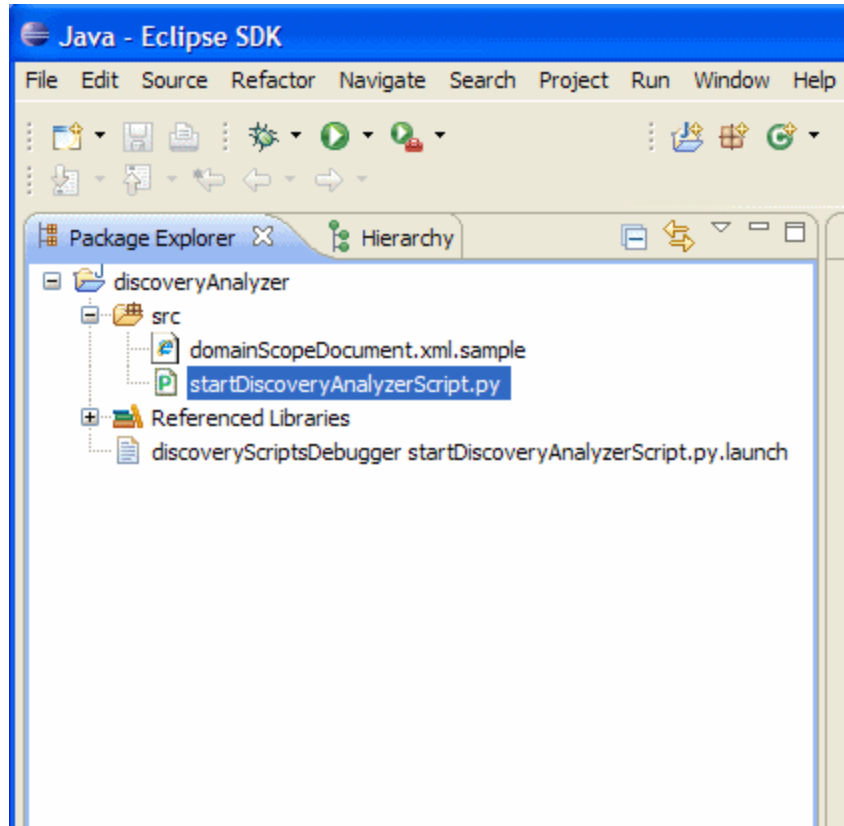


- g. Click **Apply** and then **OK**.

The Jython interpreter is now the same as the one the Probe is using.

7. Run Discovery Analyzer

- a. Add a breakpoint in the Jython script to be debugged.
- b. To start Discovery Analyzer, select **startDiscoveryAnalyzerScript.py** in the **discoveryAnalyzerWorkspace\src** project. Right-click the file and choose **Debug as > Jython run**.

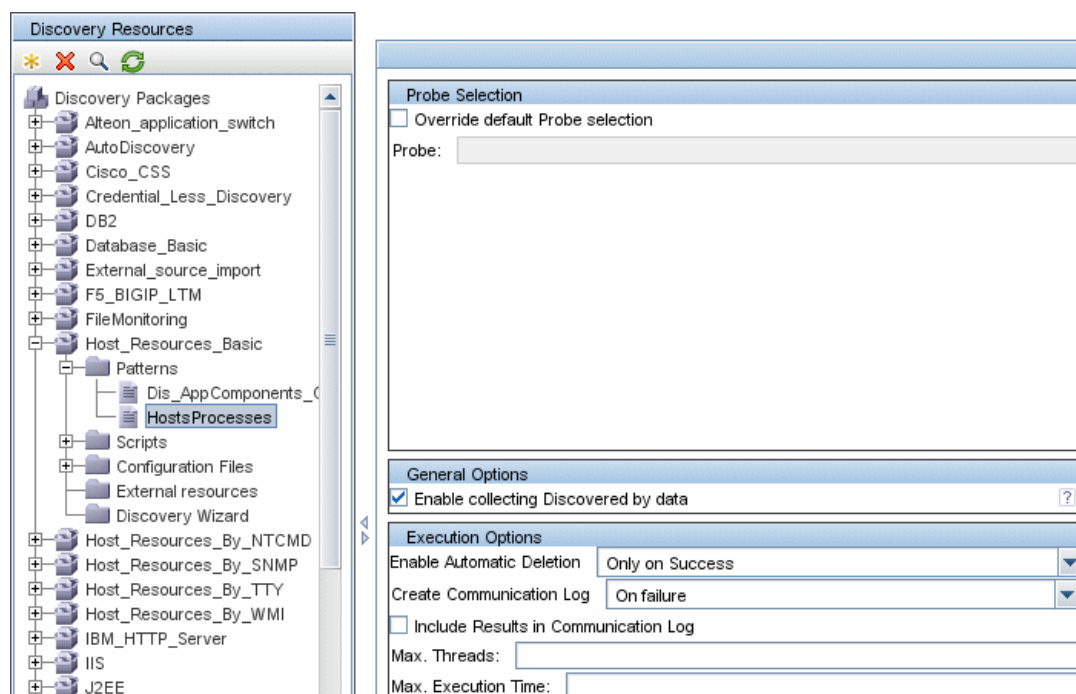


Record DFM Code

It can be very useful to record an entire execution, including all parameters, for example, when debugging and testing code. This task describes how to record an entire execution with all relevant variables. Furthermore, you can view extra debug information that is usually not printed to log files even at the debug level.

To record DFM code:

1. Access **Admin > RTSM Administration > Data Flow Management > Discovery Control Panel**. Right-click the job whose run must be logged and select **Edit adapter** to open the Adapter Management application.
2. Locate the **Execution Options** pane in the **Adapter Configuration** tab, as shown below.



3. Change the **Create communication log** box to **Always**. For details on setting logging options, see "Execution Options Pane" in the Data Flow Management Guide.

The following example is the XML log file that is created when the `Host Connection by Shell` job is run and the **Create communication logs** box is set to **Always** or **On Failure**:

Job name	Trigger CI data
<pre>- <execution jobId="Host Connection by Shell" destinationid="0e9787433d65e4a68839bfa8b224c92d"> - <destination> <destinationData name="ip_domain">DefaultDomain</destinationData> <destinationData name="hostId" /> <destinationData name="ip_address">16.59.63.34</destinationData> <destinationData name="id">0e9787433d65e4a68839bfa8b224c92d</destinationData> </destination></pre>	

The following example shows the message and stacktrace parameters:

Stacktrace

```
- <exec start="18:41:55" duration="2062" type="ssh" credentialsId="f464999bdf5a1e1407b479b6f730d5b">
  <cmd>[CDATA: client_connect]</cmd>
  <result IS_NULL="Y" />
- <error class="com.hp.ucmdb.discovery.probe.services.dynamic.agents.SSHAgentException">
  <message>[CDATA: Failed to connect: Error connecting: Connection refused: connect]</message>
  - <stacktrace>
    <frame class="com.hp.ucmdb.discovery.probe.services.dynamic.agents.SSHAgent" method="connect" file="SSHAgent.java" line="100">
    <frame class="com.hp.ucmdb.discovery.probe.clients.shell.SSHClient" method="createWrapper" file="SSHClient.java" line="100">
    <frame class="com.hp.ucmdb.discovery.probe.clients.BaseClient" method="initPrivate" file="BaseClient.java" line="100">
```


Jython Libraries and Utilities

Several utility scripts are used widely in adapters. These scripts are part of the `AutoDiscovery` package and are located under:

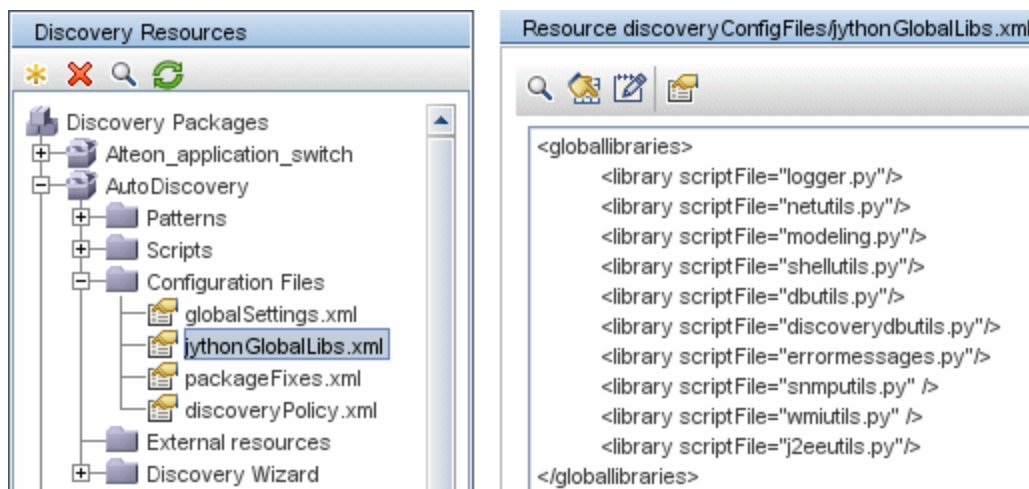
C:\hp\UCMDB\DataFlowProbe\runtime\probeManager\discoveryScripts with the other scripts that are downloaded to the Probe.

Note: The `discoveryScript` folder is created dynamically when the Probe begins working.

To use one of the utility scripts, add the following import line to the import section of the script:

```
import <script name>
```

The `AutoDiscovery` Python library contains Jython utility scripts. These library scripts are considered DFM's external library. They are defined in the `jythonGlobalLibs.xml` file (located in the **Configuration Files** folder).



Each script that appears in the `jythonGlobalLibs.xml` file is loaded by default at Probe startup, so there is no need to use them explicitly in the adapter definition.

This section includes the following topics:

- "logger.py" below
- "modeling.py" on the next page
- "netutils.py" on the next page
- "shellutils.py" on the next page

logger.py

The **logger.py** script contains log utilities and helper functions for error reporting. You can call its debug, info, and error APIs to write to the log files. Log messages are recorded in **C:\hp\UCMDB\DataFlowProbe\runtime\log**.

Messages are entered in the log file according to the debug level defined for the `PATTERNS_DEBUG` appender in the **C:\hp\UCMDB\DataFlowProbe\conf\log\probeMgrLog4j.properties** file. (By default, the level is `DEBUG`.) For details, see "Error Severity Levels" on page 97.

```
#####  
##### PATTERNS_DEBUG log  
#####  
#####  
#####  
log4j.category.PATTERNS_DEBUG=DEBUG, PATTERNS_DEBUG  
log4j.appender.PATTERNS_DEBUG=org.apache.log4j.RollingFileAppender  
log4j.appender.PATTERNS_  
DEBUG.File=C:\hp\UCMDB\DataFlowProbe\runtime\log\probeMgr-  
patternsDebug.log  
log4j.appender.PATTERNS_DEBUG.Append=true  
log4j.appender.PATTERNS_DEBUG.MaxFileSize=15MB  
log4j.appender.PATTERNS_DEBUG.Threshold=DEBUG  
log4j.appender.PATTERNS_DEBUG.MaxBackupIndex=10  
log4j.appender.PATTERNS_DEBUG.layout=org.apache.log4j.PatternLayout  
log4j.appender.PATTERNS_DEBUG.layout.ConversionPattern=<%d> [%-5p]  
[%t] - %m%n  
log4j.appender.PATTERNS_DEBUG.encoding=UTF-8
```

The info and error messages also appear in the Command Prompt console.

There are two sets of APIs:

- `logger.<debug/info/warn/error>`
- `logger.<debugException/infoException/warnException/errorException>`

The first set issues the concatenation of all its string arguments at the appropriate log level and the second set issues the concatenation as well as issuing the stack trace of the most recently-thrown exception, to provide more information, for example:

```
logger.debug('found the result')  
logger.errorException('Error in discovery')
```

modeling.py

The **modeling.py** script contains APIs for creating hosts, IPs, process CIs, and so on. These APIs enable the creation of common objects and make the code more readable. For example:

```
ipOSH= modeling.createIpOSH(ip)  
host = modeling.createHostOSH(ip_address)  
member1 = modeling.createLinkOSH('member', ipOSH, networkOSH)
```

netutils.py

The **netutils.py** library is used to retrieve network and TCP information, such as retrieving operating system names, checking if a MAC address is valid, checking if an IP address is valid, and so on. For example:

```
dnsName = netutils.getHostName(ip, ip)  
isValidIp = netutils.isValidIp(ip_address)  
address = netutils.getHostAddress(hostName)
```

shellutils.py

The **shellutils.py** library provides an API for executing shell commands and retrieving the end status of an executed command, and enables running multiple commands based on that end

status. The library is initialized with a Shell Client, and uses the client to run commands and retrieve results. For example:

```
ttyClient = clientFactory.createClient(Props)
clientShUtils = shellutils.ShellUtils(ttyClient)
if (clientShUtils.isWinOs()):
    logger.debug ('discovering Windows..')
```

Chapter 4

Error Messages

This chapter includes:

Error Messages Overview	93
Error-Writing Conventions	94
Error Severity Levels	97

Error Messages Overview

During discovery, many errors may be uncovered, for example, connection failures, hardware problems, exceptions, time-outs, and so on. DFM displays these errors in Discovery Control Panel, in both Basic and Advanced Mode, whenever the regular discovery flow does not succeed. You can drill down from the Trigger CI that caused the problem to view the error message itself.

DFM differentiates between errors that can sometimes be ignored (for example, an unreachable host) and errors that must be dealt with (for example, credentials problems or missing configuration or DLL files). Moreover, DFM reports errors once, even if the same error occurs on successive runs, and reports an error even if it occurs once only.

When creating a package, you can add appropriate messages as resources to the package. During package deployment, the messages are also deployed in the correct location. Messages must conform to conventions, as described in ["Error-Writing Conventions" on the next page](#).

DFM supports multi-language error messages. You can localize the messages you write so that they appear in the local language.

For details on searching for errors, see "Discovery Overview/Status Pane" in the *Data Flow Management Guide*.

For details on setting communication logs, see "Execution Options Pane" in the *Data Flow Management Guide*.

Error-Writing Conventions

- Each error is identified by an error message code and an array of arguments (**int**, **String[]**). A combination of a message code and an array of arguments defines a specific error. The array of parameters can be null.
- Each error code is mapped to a **short message** which is a fixed string and a **detailed message** which is a template string contains zero or more arguments. Matching is assumed between the number of arguments in the template and the actual number of parameters.

Example of Error Message Code:

10234 may represent an error with the short message:

```
Connection Error
```

and the detailed message:

```
Could not connect via {0} protocol due to timeout of {1} msec
```

where

{0} = the first argument: a protocol name

{1} = the second argument: the timeout length in msec

This section also includes the following topics:

- ["Property File Content" below](#)
- ["Error Messages Property File" below](#)
- ["Locale Naming Conventions" on the next page](#)
- ["Error Message Codes" on the next page](#)
- ["Unclassified Content Errors" on the next page](#)
- ["Changes in Framework" on page 96](#)

Property File Content

A property file should contain two keys for each error message code. For example, for error 45:

- **DDM_ERROR_MESSAGE_SHORT_45**. Short error description.
- **DDM_ERROR_MESSAGE_LONG_45**. Long error description (can contain parameters, for example, **{0},{1}**).

Error Messages Property File

A property file contains a map between an error message code and two messages (short and detailed).

Once a property file is deployed, its data is merged with existing data, that is, new message codes are added while old message codes are overridden.

Infrastructure property files are part of the **AutoDiscoveryInfra** package.

Locale Naming Conventions

- For the default locale: **<file name>.properties.errors**
- For a specific locale: **<file name>_xx.properties.errors**

where **xx** is the locale (for example, **infraerr_fr.properties.errors** or **infraerr_en_us.properties.errors**).

Error Message Codes

The following error codes are included by default with HP Universal CMDB. You can add your own error messages to this list.

Error Name	Error Code	Description
Internal	100-199	Mostly resolved from exceptions thrown during Jython script runs
Connection	200-299	Connection failed, no agent on target machine, destination unreachable, and so on
Credential Related	300-399	Permission denied, connection attempt blocked due to a lack of credentials
Timeout	400-499	Time-out during connection/command
Unexpected or Invalid Behavior	500-599	Missing configuration files, unexpected interruptions, and so on
Information Retrieval	600-699	Missing information on target machines, failure querying agent for information, and so on
Resources Related	700-799	Errors relating to out-of-memory or clients not released properly
Parsing	800-899	Error parsing text
Encoding	900	Error in input, unsupported encoding
SQL Related	901-903, 924	Errors received from SQL operations
HTTP Related	904-909	Errors generated during HTTP connections, parsed from HTTP error codes.
Specific Application	910-923	Error reported due to application-specific problems, for example, wrong LSOF version, No Queue Managers found, and so on

Unclassified Content Errors

To support old content without causing a regression, the application and SDK relevant methods handle errors of message code 100 (that is, unclassified script error) differently.

These errors are not grouped (that is, they are not considered as being errors of the same type) by their message code but are grouped by the content of the message. That is, if a script reports an error by the old, deprecated methods (with a message string and without an error code), all messages receive the same error code, but in the application or in the SDK relevant methods, different messages are displayed as different errors.

Changes in Framework

(com.hp.ucmdb.discovery.library.execution.BaseFramework)

The following methods are added to the interface:

- `void reportError(int msgCode, String[] params);`
- `void reportWarning(int msgCode, String[] params);`
- `void reportFatal(int msgCode, String[] params);`

The following old methods are still supported for backward compatibility purposes but are marked as deprecated:

- `void reportError(String message);`
- `void reportWarning (String message);`
- `void reportFatal (String message);`

Error Severity Levels

When an adapter finishes running against a trigger CI, it returns a status. If no error or warning is reported, the status is **Success**.

Severity levels are listed here from the narrowest to widest scope:

Fatal Errors

This level reports serious errors such as a problem with the infrastructure, missing DLL files, or exceptions:

- Failed generating the task (Probe is not found, variables are not found, and so on)
- It is not possible to run the script
- Processing of the results fails on the Server and the data is not written to the CMDB

Errors

This level reports problems that cause DFM not to retrieve data. Look through these errors as they usually require some action to be taken (for example, to increase time-out, to change a range, to change a parameter, to add another user credential, and so on).

- In cases where user intervention may help, an error is reported, either a credentials or network problem that may need further investigation. (These are not errors in discovery but in configuration.)
- Internal failure, usually because of unexpected behavior from the discovered machine or application, for example, missing configuration files, and so on

Warning

When a run is successful but there may be non-serious problems that you should be aware of, DFM marks the severity as **Warning**. You should look at these CIs to see whether data is missing, before beginning a more detailed debugging session. **Warning** can include messages about the lack of an installed agent on a remote host, or that invalid data caused an attribute not to be properly calculated.

- Missing connection agent (SNMP, WMI)
- Discovery succeeds, but not all available information is discovered

Chapter 5

Developing Generic Database Adapters

This chapter includes:

Generic Database Adapter Overview	99
TQL Queries for the Generic Database Adapter	100
Reconciliation	102
Hibernate as JPA Provider	103
Prepare for Adapter Creation	106
Prepare the Adapter Package	111
Upgrade the Generic DB Adapter from 9.00 or 9.01 to 9.02 and Later	114
Configure the Adapter – Minimal Method	115
Configure the Adapter – Advanced Method	118
Implement a Plugin	123
Deploy the Adapter	126
Edit the Adapter	127
Create an Integration Point	128
Create a View	129
Calculate the Results	130
View the Results	131
View Reports	132
Enable Log Files	133
Use Eclipse to Map Between CIT Attributes and Database Tables	134
Adapter Configuration Files	140
Out-of-the-Box Converters	158
Plugins	162
Configuration Examples	163
Adapter Log Files	172
External References	174
Troubleshooting and Limitations	175

Generic Database Adapter Overview

The purpose of the generic database adapter platform is to create adapters that can integrate with relational database management systems (RDBMS) and run TQL queries and population jobs against the database. The RDBMS supported by the generic database adapter are Oracle, Microsoft SQL Server, and MySQL.

This version of the database adapter implementation is based on a JPA (Java Persistence API) standard with the Hibernate ORM library as the persistence provider.

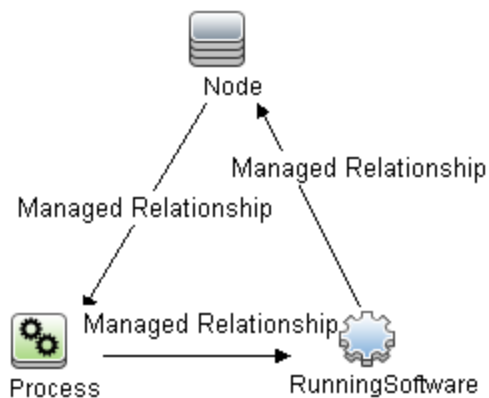
TQL Queries for the Generic Database Adapter

For population jobs, every required layout of a CI must be checked in the Layout Settings Dialog Box in the Modeling Studio. For details, see Query Node/Relationship Properties Dialog Box in the *Modeling Guide*. It is important to note that a CI might require an attribute to be identified, and without those attributes the CI will fail to be added to RTSM.

The following limitations exist on the TQL queries calculated by the Generic Database Adapter only:

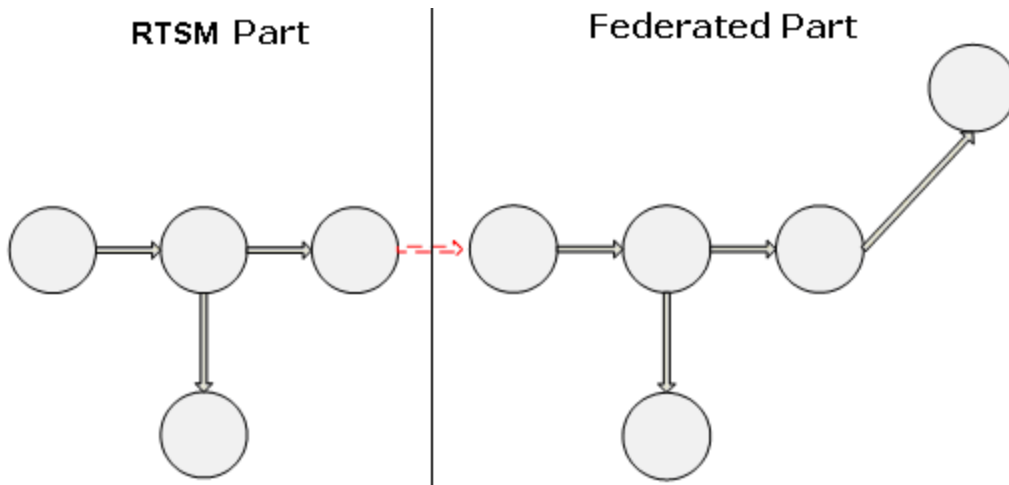
- Subgraphs are not supported
- Compound relationships are not supported
- Cycles or cycle parts are not supported

The following TQL query is an example of a cycle:



- Function layout is not supported.
- 0..0 cardinality is not supported.
- The `Join` relationship is not supported.
- Qualifier conditions are not supported.
- To connect between two CIs, a relationship in the form of a table or foreign key must exist in the

external database source.



Reconciliation

Reconciliation is carried out as part of the TQL calculation on the adapter side. For reconciliation to occur, the RTSM side is mapped to a federated entity called reconciliation CIT.

Mapping. Each attribute in the RTSM is mapped to a column in the data source.

Although mapping is done directly, transformation functions on the mapping data are also supported. You can add new functions through the Java code (for example, lowercase, uppercase). The purpose of these functions is to enable value conversions (values that are stored in the RTSM in one format and in the federated database in another format).

Note:

- To connect the RTSM and external database source, an appropriate association must exist in the database. For details, see ["Prerequisites" on page 106](#).
- Reconciliation with the RTSM `id` is also supported.

Hibernate as JPA Provider

Hibernate is an object-relational (OR) mapping tool, which enables mapping Java classes to tables over several types of relational databases (for example, Oracle and Microsoft SQL Server). For details, see ["Functional Limitations" on page 175](#).

In an elementary mapping, each Java class is mapped to a single table. More advanced mapping enables inheritance mapping (as can occur in the RTSM database).

Other supported features include mapping a class to several tables, support for collections, and associations of types one-to-one, one-to-many, and many-to-one. For details, see ["Associations" on the next page](#) below.

For our purposes, there is no need to create Java classes. The mapping is defined from the RTSM class model CITs to the database tables.

This section also includes the following topics:

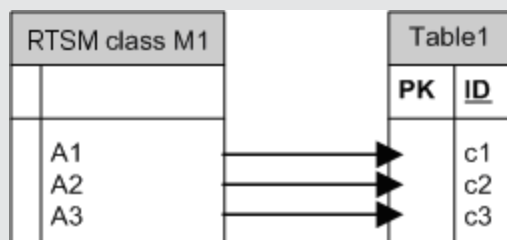
- ["Examples of Object-Relational Mapping" below](#)
- ["Associations" on the next page](#)
- ["Usability" on page 105](#)

Examples of Object-Relational Mapping

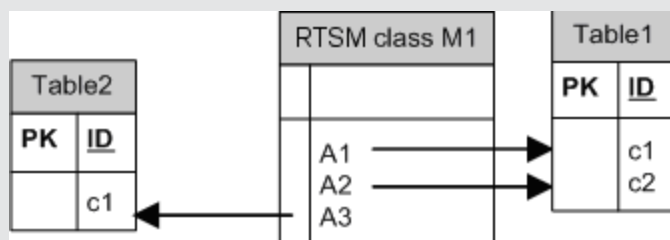
The following examples describe object-relational mapping:

Example of One RTSM Class Mapped to One Database Table:

Class M1, with attributes A1, A2, and A3, is mapped to table 1 columns c1, c2, and c3. This means that any M1 instance has a matching row in table 1.

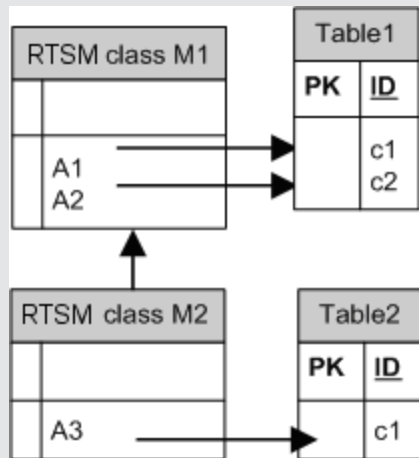


Example of One RTSM Class Mapped to Two Database Tables:



Example of Inheritance:

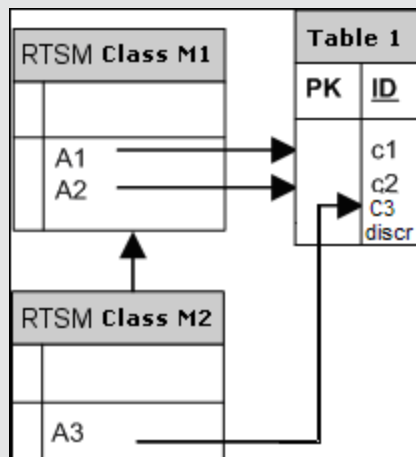
This case is used in the RTSM, where each class has its own database table.



Example of Single Table Inheritance with Discriminator:

An entire hierarchy of classes is mapped to a single database table, whose columns comprise a super-set of all attributes of the mapped classes. The table also contains an additional column (*Discriminator*), whose value indicates which specific class should be mapped to this entry.

When you use discriminator capabilities, you cannot skip a class in the hierarchy; that is, since C3 inherits from C2 and C2 inherits from C1, you cannot just define C1 and C3, you must define all three classes.



Associations

There are three types of associations: one-to-many, many-to-one and many-to-many. To connect between the different database objects, one of these associations must be defined by using a foreign key column (for the one-to-many case) or a mapping table (for the many-to-many case).

Usability

As the JPA schema is very extensive, a streamlined XML file is provided to ease definitions.

The use case for using this XML file is as follows: Federated data is modeled into one federated class. This class has many-to-one relations to a non-federated RTSM class. In addition, there is only one possible relation type between the federated class and the non-federated class.

Prepare for Adapter Creation

This task describes the preparations that are necessary for creating an adapter.

Note: You can view samples for the Generic DB adapter in the UCMDb API. Specifically, the DDMi Adapter sample contains a complicated **orm.xml** file, as well as the implementations for some plugin interfaces.

This task includes the following steps:

- ["Prerequisites" below](#)
- ["Create a CI Type" on the next page](#)
- ["Create a Relationship" on page 108](#)

1. Prerequisites

To validate that you can use the database adapter with your database, check the following:

- The reconciliation classes and their attributes (also known as multinodes) exist in the database. For example, if the reconciliation is run by node name, verify that there is a table that contains a column with node names. If the reconciliation is run according to node `cmdb_id`, verify that there is a column with RTSM IDs that matches the RTSM IDs of the nodes in the RTSM. For details on reconciliation, see ["Reconciliation" on page 102](#).

ID	NAME	IP_ADDRESS
31	BABA	16.59.33.60
33	ext3.devlab.ad	16.59.59.116
46	LABM1MAM15	16.59.58.188
72	cert-3-j2ee	16.59.57.100
102	labm1sun03.devlab.ad	16.59.58.45
114	LABM2PCOE73	16.59.66.79
116	CUT	16.59.41.214
117	labm1hp4.devlab.ad	16.59.60.182

- To correlate two CITs with a relationship, there must be correlation data between the CIT tables. The correlation can be either by a foreign key column or by a mapping table. For example, to correlate between node and ticket, there must be a column in the ticket table that contains the node ID, a column in the node table with the ticket ID that is connected to it, or a mapping table whose `end1` is the node ID and `end2` is the ticket ID. For details on correlation data, see ["Hibernate as JPA Provider" on page 103](#).

The following table shows the foreign key `NODE_ID` column:

NODE_ID	CARD_ID	CARD_TYPE	CARD_NAME
2015	1	Serial Bus Controller	Intel 82801EB USB Universal Host Controller
3581	2	System	Intel 631xESB/6321ESB/3100 Chipset LPC
3581	3	Display	ATI ES1000
3581	4	Base System Peripheral	HP ProLiant iLO 2 Legacy Support Function

- Each CIT can be mapped to one or more tables. To map one CIT to more than one table, check that there is a primary table whose primary key exists in the other tables, and is a unique value column.

For example, a ticket is mapped to two tables: `ticket1` and `ticket2`. The first table has columns `c1` and `c2` and the second table has columns `c3` and `c4`. To enable them to be considered as one table, both must have the same primary key. Alternatively, the first table primary key can be a column in the second table.

In the following example, the tables share the same primary key called `CARD_ID`:

CARD_ID	CARD_TYPE	CARD_NAME
1	Serial Bus Controller	Intel 82801EB USB Universal Host Controller
2	System	Intel 631xESB/6321ESB/3100 Chipset LPC
3	Display	ATI ES1000
4	Base System Peripheral	HP ProLiant iLO 2 Legacy Support Function

CARD_ID	CARD_VENDOR
1	Hewlett-Packard Company
2	(Standard USB Host Controller)
3	Hewlett-Packard Company
4	(Standard system devices)
5	Hewlett-Packard Company

2. Create a CI Type

In this step you create a CIT that represents the data in the RDBMS (the external data source).

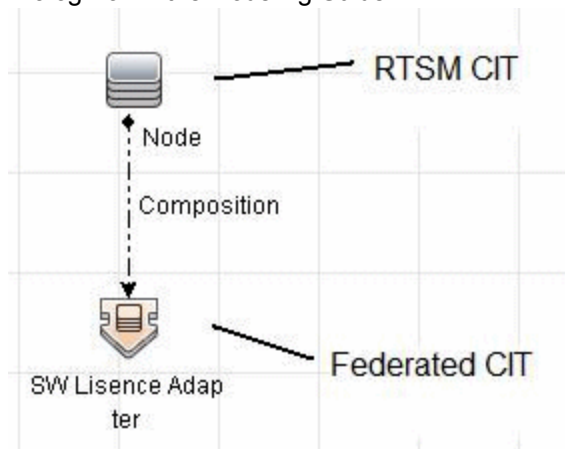
- In BSM, access the CI Type Manager and create a new CI Type. For details, see [Create a CI Type in the Modeling Guide](#).
- Add the necessary attributes to the CIT, such as last access time, vendor, and so on.

These are the attributes that the adapter will retrieve from the external data source and bring into RTSM views.

3. Create a Relationship

In this step you add a relationship between the BSM CIT and the new CIT that represents the data from the external data source.

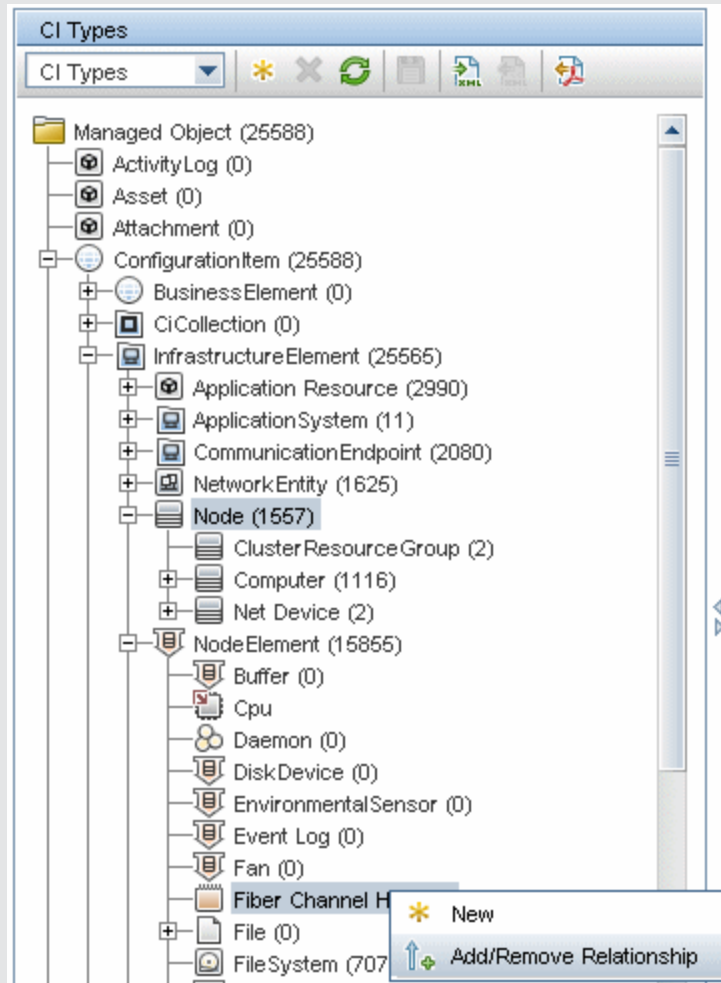
Add appropriate, valid relationships to the new CIT. For details, see Add/Remove Relationship Dialog Box in the Modeling Guide.



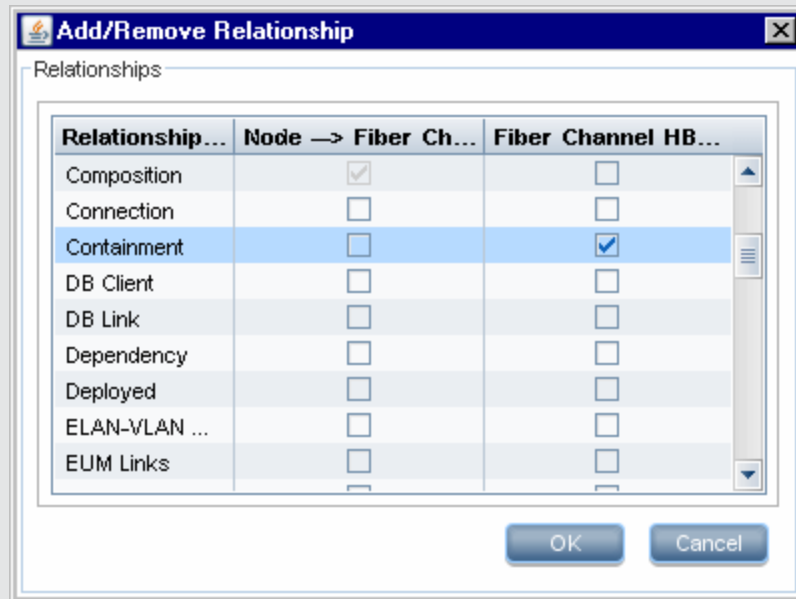
Note: At this stage, you cannot yet view the federated data or populate the external data, as you have not yet defined the method for bringing in the data.

Example of Creating a Containment Relationship:

- a. In the CIT Manager, select the two CITs:



- b. Create a **Containment** relationship between the two CITs:



Prepare the Adapter Package

In this step, you locate and configure the Generic DB adapter package.

1. Locate the **db-adapter.zip** package in the
 <HP BSM root directory>\
 odb\content\adapters folder.
2. Extract the package to a local temporary directory.
3. Edit the adapter XML file:
 - Open the **discoveryPatterns\db_adapter.xml** file in a text editor.
 - Locate the **adapter id** attribute and replace the name:

```
<pattern id="MyAdapter" displayLabel="My Adapter"
xsi:noNamespaceSchemaLocation="../../../Patterns.xsd"
description="Discovery Pattern Description"
      schemaVersion="9.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
displayName="UCMDB API Population">
```

If the adapter supports population, the following capability should be added to the **<adapter-capabilities>** element:

```
<support-replicatioin-data>
  <source>
    <changes-source>
  </source>
</support-replicatioin-data>
```

The display label or ID appears in the list of adapters in the Integration Point pane in BSM.

If the plug-in for **FcmdbPluginForSyncGetChangesTopology** has not been implemented, only the following should be added:

```
<support-replicatioin-data>
  <source>
    <!--<changes-source>-->
  </source>
</support-replicatioin-data>
```

This will return the full topology and perform auto-delete according to the returned CIs.

For details about populating the RTSM with data, see "Integration Studio Page " on page 1 of the *Data Flow Management Guide*.

- If the adapter is using the mapping engine from version 8.x (meaning that it is not using the new reconciliation mapping engine), replace the following element:

```
<default-mapping-engine>
```

with:

```
<default-mapping-  
engine>com.hp.ucmdb-  
.federation.mappingEngine.AdapterMappingEngine</default-mapping-  
engine>
```

To revert to the new mapping engine, return the element to the following value:

```
<default-mapping-engine>
```

- Locate the **category** definition:

```
<category>Generic</category>
```

Change the **Generic** category name to the category of your choice.

Note: Adapters whose categories are specified as **Generic** are not listed in the Integration Studio when you create a new integration point.

- The connection to the database can be described using a user name (schema), password, database type, database host machine name, and database name or SID.

For this type of connection, parameters have the following elements in the **parameter** section of the adapter's XML file:

```
<parameters>  
  <!--The description attribute may be written in simple text  
or HTML.-->  
  <!--The host attribute is treated as a special case by UCMDB-  
-->  
  <!--and will automatically select the probe name (if  
possible)-->  
  <!--according to this attribute's value.-->  
  <!--Display name and description may be overwritten by I18N  
values-->  
    <parameter name="host" display-name="Hostname/IP"  
type="string" description="The host name or IP address of the  
remote machine" mandatory="false" order-index="10" />  
    <parameter name="port" display-name="Port" type="integer"  
description="The remote machine's connection port"  
mandatory="false" order-index="11" />  
    <parameter name="dbtype" display-name="DB Type"  
type="string" description="The type of database" valid-  
values="Oracle;SQLServer;MySQL;BO" mandatory="false" order-  
index="13">Oracle</parameter>  
    <parameter name="dbname" display-name="DB Name/SID"  
type="string" description="The name of the database or its SID  
(in case of Oracle)" mandatory="false" order-index="13" />  
    <parameter name="credentialsId" display-name="Credentials  
ID" type="integer" description="The credentials to be used"  
mandatory="true" order-index="12" />  
</parameters>
```


Note: This is the default configuration. Therefore, the **db_adapter.xml** file, already contains this definition.

There are situations in which the connection to the database cannot be configured in this way. For example, connecting to Oracle RAC or connecting using a database driver other than the one supplied with the RTSM.

For these situations, you can describe the connection using user name (schema), password, and a connection URL string.

To define this, edit the adapter's XML parameters section as follows:

```
<parameters>
  <!--The description attribute may be written in simple text or
  HTML.-->
  <!--The host attribute is treated as a special case by
  CMDBERTSM-->
  <!--and will automatically select the probe name (if possible)
  -->
  <!--according to this attribute's value.-->
  <!--Display name and description may be overwritten by I18N
  values-->
  <parameter name="url" display-name="Connection String"
  type="string" description="The connection string to connect to the
  database" mandatory="true" order-index="10" />
  <parameter name="credentialsId" display-name="Credentials
  ID" type="integer" description="The credentials to be used"
  mandatory="true" order-index="12" />
</parameters>
```

An example of a URL that connects to an Oracle RAC using the out-of-the- box Data Direct driver is:

jdbc:mercury:oracle://labm3amdb17:1521;ServiceName=RACQA;AlternateServers=(labm3amdb18:1521);LoadBalancing=true.

4. In the temporary directory, open the **adapterCode** folder and rename **GenericDBAdapter** to the value of **adapter id** that was used in the previous step.

This folder contains the adapter's configuration, for example, the adapter name, the queries and classes in the RTSM, and the fields in the RDBMS that the adapter supports.

5. Configure the adapter as required. For details, see ["Configure the Adapter – Minimal Method" on page 115](#).
6. Create a *.zip file with the same name as you gave to the **adapter id** attribute, as described in the step 3 above.

Note: The **descriptor.xml** file is a default file that exists in every package.

7. Save the new package that you created in the previous step. The default directory for adapters is: **<HP BSM root directory>\odb\content\adapters**.

Upgrade the Generic DB Adapter from 9.00 or 9.01 to 9.02 and Later

1. Copy your adapter package to a local temporary directory.
2. Extract the files.
3. Remove the following files from the **adapterCode\<Your Adapter Name>** folder:
 - **asm.jar**
 - **asm-attrs.jar**
 - **cglib.jar**
 - **db-adapter.jar**
 - **jboss-archive-browsing.jar**
 - **saxon-b.jar**
4. Recreate your adapter package.

Note: For any deployed Generic DB adapters that you may have, the UCMDB installer will remove the necessary files from the UCMDB and Probe file system. However, you still need to fix the package yourself, in order to re-deploy it when necessary.

Configure the Adapter – Minimal Method

The following procedure describes a method of mapping the class model in the RTSM to an RDBMS.

These configuration files are located in the **db-adapter.zip** package in the **<HP BSM root directory>\odb\content\adapters** folder that you extracted in the step ["Extract the package to a local temporary directory."](#) on page 111 in ["Prepare the Adapter Package"](#) on page 111.

Note: The **orm.xml** file that is automatically generated as a result of running this method is a good example that you can use when working with the advanced method.

You would use this minimal method when you need to:

- Federate/populate a single node such as a node attribute.
- Demonstrate the Generic Database Adapter capabilities.

This method:

- supports one-node federation\population only
- supports many-to-one virtual relationships only

This task includes the following steps:

- ["Configure the adapter.conf File" below](#)
- ["Configure the simplifiedConfiguration.xml File" below](#)

Configure the adapter.conf File

In this step, you change the settings in the `adapter.conf` file so that the adapter uses the simplified configuration method.

1. Open the **adapter.conf** file in a text editor.
2. Locate the following line: **use.simplified.xml.config=<true/false>**.
3. Change it to **use.simplified.xml.config=true**.

Configure the simplifiedConfiguration.xml File

In this step, you configure the **simplifiedConfiguration.xml** file by mapping the CIT in the RTSM to the fields in the RDBMS table.

1. Open the **simplifiedConfiguration.xml** file in a text editor.

This file includes a template that you use for each entity to be mapped.

Note: Do not edit the **simplifiedConfiguration.xml** file in any version of Notepad from Microsoft Corporation. Use Notepad++, UltraEdit, or some other third-party text editor.

2. Make changes to the following attributes:
 - The CIT name in BSM (cmdb-class-name) and the corresponding table name in the RDBMS (default-table-name):

```
<cmdb-class cmdb-class-name="node" default-table-name="Device">
```

The `cmdb-class-name` attribute is taken from the node CIT:



The `default-table-name` attribute is taken from the Device table:

	Column Name	Data Type	Length	Allow Nulls
1	Device_ID	int		<input type="checkbox"/>
2	Device_Discovered	enum		<input type="checkbox"/>
3	Device_ManagedCategory	enum		<input checked="" type="checkbox"/>
4	Device_PreferredMACAddress	varchar	12	<input checked="" type="checkbox"/>
5	Device_PreferredIPAddress	varchar	15	<input checked="" type="checkbox"/>
6	Device_LogicalSubNet	varchar	50	<input checked="" type="checkbox"/>
7	Device_Tag	text		<input checked="" type="checkbox"/>
8	Device_Label	varchar	255	<input checked="" type="checkbox"/>
9	DeviceCategory_ID	int		<input checked="" type="checkbox"/>
10	DeviceIcon_ID	int		<input checked="" type="checkbox"/>
11	Device_Description	text		<input checked="" type="checkbox"/>
12	Device_ObjectID	text		<input checked="" type="checkbox"/>
13	Device_Contact	text		<input checked="" type="checkbox"/>
14	Device_Name	text		<input checked="" type="checkbox"/>
15	Device_Location	text		<input checked="" type="checkbox"/>
16	Device_NetBIOS	varchar	255	<input checked="" type="checkbox"/>

- The unique identifier in the RDBMS:

```
<primary-key column-name="Device_ID" />
```

- The reconciliation rule (reconciliation-by-two-nodes):

```
<reconciliation-by-two-nodes connected-node-cmdb-class-name="ip_
address" cmdb-link-type="containment">
```

- The reconciliation attribute in BSM (`cmdb-attribute-name`) and in the RDBMS (`column-name`):

```
<connected-node-attribute cmdb-attribute-name="name" column-
name="[column_name]" />
```

- The name of the CIT (`cmdb-class-name`) and the name of the corresponding table in the RDBMS (`default-table-name`). Also the RTSM relationship (`connected-cmdb-class-name`) and the CIT relationship (`link-class-name`):

```
<class cmdb-class-name="sw_sub_component" default-table-
name="SWSubComponent" connected-cmdb-class-name="node" link-
class-name="composition">
```

- The primary key and the foreign key:

```
<foreign-primary-key column-name="Device_ID" cmdb-class-primary-  
key-column="Device_ID" />
```

- The unique identifier in the RDBMS:

```
<primary-key column-name="Device_ID" />
```

- The mapping between the RTSM attribute (cmdb-attribute-name) and the column name in the RDBMS (column-name):

```
<attribute cmdb-attribute-name="last_access_time" column-  
name="SWSubComponent_LastAccess TimeStamp" />
```

3. Save the file.

Configure the Adapter – Advanced Method

These configuration files are located in the **db-adapter.zip** package in the **<HP BSM root directory>\odb\content\adapters** folder that you extracted in the step "Extract the package to a local temporary directory." on page 111 in "Prepare the Adapter Package" on page 111.

This task includes the following steps:

- "Configure the orm.xml File" below
- "Configure the reconciliation_types.txt File" on page 121
- "Configure the reconciliation_rules.txt File" on page 121

Configure the orm.xml File

In this step, you map the CITs and relationships in the RTSM to the tables in the RDBMS.

1. Open the **orm.xml** file in a text editor.

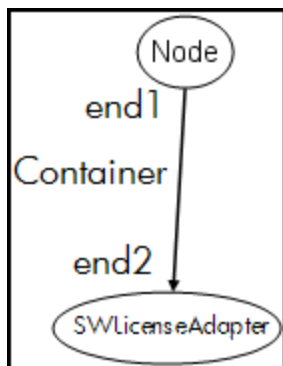
This file, by default, contains a template that you use to map as many CITs and relationships as needed.

Note: Do not edit the **orm.xml** file in any version of Notepad from Microsoft Corporation. Use Notepad++, UltraEdit, or some other third-party text editor.

2. Make changes to the file according to the data entities to be mapped. For details, see the following examples.

The following types of relationships may be mapped in the **orm.xml** file:

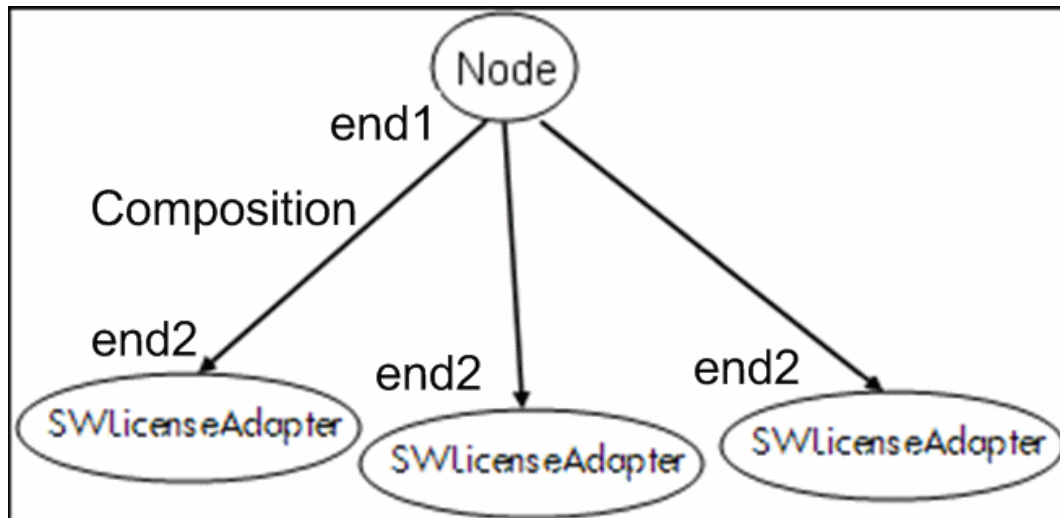
- One to one:



The code for this type of relationship is:

```
<one-to-one name="end1" target-entity="node">
    <join-column name="Device_ID" >
</one-to-one>
<one-to-one name="end2" target-entity="sw_sub_component">
    <join-column name="Device_ID" >
    <join-column name="Version_ID" >
</one-to-one>
```

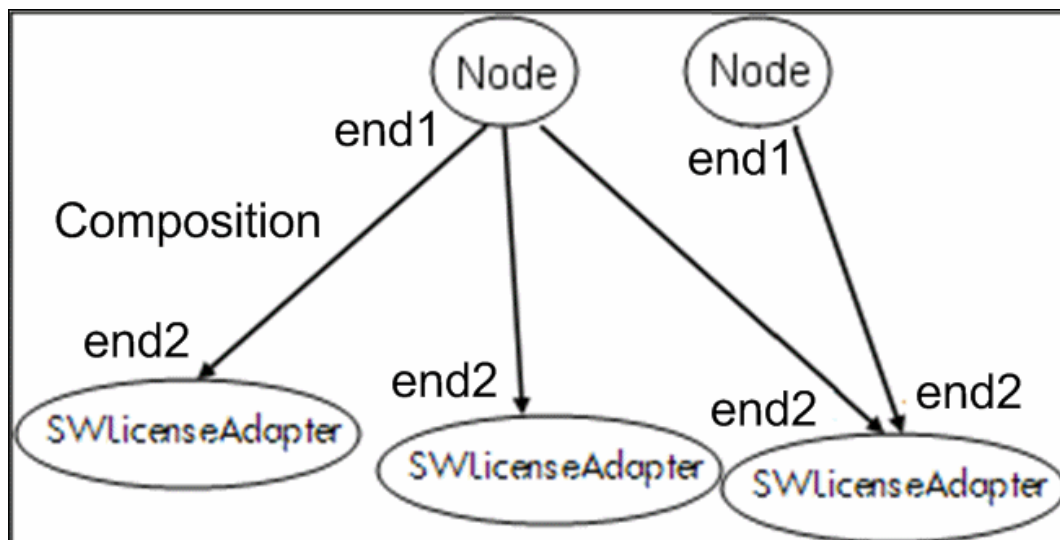
- Many to one:



The code for this type of relationship is:

```
<many-to-one name="end1" target-entity="node">
    <join-column name="Device_ID" >
</many-to-one>
<one-to-one name="end2" target-entity="sw_sub_component">
    <join-column name="Device_ID" >
    <join-column name="Version_ID" >
</one-to-one>
```

- Many to many:



The code for this type of relationship is:

```
<many-to-one name="end1" target-entity="node">
    <join-column name="Device_ID" >
</many-to-one>
<many-to-one name="end2" target-entity="sw_sub_component">
```

```
<join-column name="Device_ID" >  
  <join-column name="Version_ID" >  
</many-to-one>
```

For details about naming conventions, see ["Naming Conventions" on page 147](#).

Example of Entity Mapping Between the Data Model and the RDBMS:

Note: Attributes that do not have to be configured are omitted from the following examples.

- The class of the RTSM CIT:

```
<entity class="generic_db_adapter.node">
```

- The name of the table in the RDBMS:

```
<table name="Device" />
```

- The column name of the unique identifier in the RDBMS table:

```
<column name="Device ID" />
```

- The name of the attribute in the RTSM CIT:

```
<basic name="name">
```

- The name of the table field in the external data source:

```
<column name="Device_Name" />
```

- The name of the new CIT you created in ["Create a CI Type" on page 107](#):

```
<entity class="generic_db_adapter.MyAdapter">
```

- The name of the corresponding table in the RDBMS:

```
<table name="SW_License" />
```

- The unique identity in the RDBMS:

- The attribute name in the RTSM CIT and the name of the corresponding attribute in the RDBMS:

Example of Relationship Mapping Between the Data Model and the RDBMS:

- The class of the RTSM relationship:

```
<entity class="generic_db_adapter.node_containment_
MyAdapter">
```

- The name of the RDBMS table where the relationship is performed:

```
<table name="MyAdapter" />
```

- The unique ID in the RDBMS:

```
<id name="id1">
    <column updatable="false" insertable="false"
name="Device_ID">
        <generated-value strategy="TABLE" />
    </id>
<id name="id2">
    <column updatable="false" insertable="false"
name="Version_ID">
        <generated-value strategy="TABLE" />
    </id>
```

- The relationship type and the RTSM CIT:

```
<many-to-one target-entity="node" name="end1">
```

- The primary key and foreign key fields in the RDBMS:

```
<join-column updatable="false" insertable="false"
referenced-column-name="[column_name]" name="Device_ID" />
```

Configure the reconciliation_types.txt File

Open the **reconciliation_types.txt** file in a text editor.

For details, see ["The reconciliation_types.txt file" on page 152](#).

Configure the reconciliation_rules.txt File

In this step you define the rules by which the adapter reconciles the RTSM and the RDBMS (only if Mapping Engine is used, for backward compatibility with version 8.x):

1. Open **META-INF\reconciliation_rules.txt** in a text editor.
2. Make changes to the file according to the CIT you are mapping. For example, to map a node CIT, use the following expression:

```
multinode[node] ordered expression[^name]
```

Note:

- If the data in the database is case sensitive, do not delete the control character (^).
- Check that each opening square bracket has a matching closing bracket.

For details, see ["The reconciliation_rules.txt File \(for backwards compatibility\)"](#) on page 152.

Implement a Plugin

This task describes how to implement and deploy a Generic DB Adapter with plugins.

Note: Before writing a plugin for an adapter, make sure you have completed all the necessary steps in ["Prepare the Adapter Package" on page 111](#).

1. Copy the following jar files from the UCMDB server installation directory to your development class path:
 - Copy the **db-interfaces.jar** file and **db-interfaces-javadoc.jar** file from the **tools\adapter-dev-kit\db-adapter-framework** folder.
 - Copy the **federation-api.jar** file and **federation-api-javadoc.jar** file from the **tools\adapter-dev-kit\SampleAdapters\production-lib** folder.

Note: More information about developing a plugin can be found in the **db-interfaces-javadoc.jar** and **federation-api-javadoc.jar** files and in the online documentation at:

- **C:\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\DBAdapterFramework_JavaAPI\index.html**
- **C:\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\Federation_JavaAPI\index.html**

2. Write a Java class implementing the plugin's Java interface. The interfaces are defined in the **db-interfaces.jar** file. The table below specifies the interface that must be implemented for each plugin:

Plugin Type	Interface Name	Method
Synchronize Full Topology	FcmdbPluginForSyncGetFullTopology	getFullTopology
Synchronize Changes	FcmdbPluginForSyncGetChangesTopology	getChangesTopology
Synchronize Layout	FcmdbPluginForSyncGetLayout	getLayout
Retrieve Supported Queries	FcmdbPluginForSyncGetSupportedQueries	getSupportedQueries
Alter TQL query definition and results	FcmdbPluginGetTopologyCmdbFormat	getTopologyCmdbFormat
Alter layout request for CIs	FcmdbPluginGetCIsLayout	getCIsLayout

Plugin Type	Interface Name	Method
Alter layout request for links	FcmdbPluginGetRelationsLayout	getRelationsLayout

The plugin's class must have a public default constructor. Also, all of the interfaces expose a method called `initPlugin`. This method is guaranteed to be called before any other method and is used to initialize the adapter with the containing adapter's environment object.

If **FcmdbPluginForSyncGetChangesTopology** is implemented, there are two different ways to report the changes:

- **Report the entire root topology at all times.** According to this topology, the auto-delete function finds which CIs should be removed. In this case, the auto-delete function should be enabled by using the following:

```
<autoDeleteCITs isEnabled="true">
    <CIT>link</CIT>
    <CIT>object</CIT>
</autoDeleteCITs>
```

- **Report each CI instance that was removed/updated.** In this case the auto-delete mechanism should be disabled by using the following:

```
<autoDeleteCITs isEnabled="false">
    <CIT>link</CIT>
    <CIT>object</CIT>
</autoDeleteCITs>
```

3. Make sure you have the Federation SDK JAR and the Generic DB Adapter JARs in your class path before compiling your Java code. The Federation SDK is the **federation_api.jar** file, which can be found in the **<HP BSM root directory>\odb\lib** directory.
4. Pack your class into a jar file and put it under the `adapterCode\<Your Adapter Name>` folder in the adapter package, prior to deploying it.

The plug-ins are configured using the **plugins.txt** file, located in the **\META-INF** folder of the adapter.

The following is an example of the file from the DDMi adapter:

```
# mandatory plugin to sync full topology
[getFullTopology]
com.hp.ucmdb.adapters.ed.plugins.replication.EDReplicationPlugin
# mandatory plugin to sync changes in topology
[getChangesTopology]
com.hp.ucmdb.adapters.ed.plugins.replication.EDReplicationPlugin
# mandatory plugin to sync layout
[getLayout]
com.hp.ucmdb.adapters.ed.plugins.replication.EDReplicationPlugin
# plugin to get supported queries in sync. If not defined return
all tqls names
```

```
[getSupportedQueries]
# internal not mandatory plugin to change tql definition and tql
result
[getTopologyCmdFormat]
# internal not mandatory plugin to change layout request and CIs
result
[getCisLayout]
# internal not mandatory plugin to change layout request and
relations result
[getRelationsLayout]
```

Legend:



- A comment line.

[<Adapter Type>] – Start of the definition section for a specific adapter type.

There can be an empty line under each [<Adapter Type>], meaning that there is no plugin class associated, or the fully qualified name of your plugin class can be listed.

5. Pack your adapter with the new jar file and the updated **plugins.xml** file. The remainder of the files in the package should be the same as in any adapter based on the Generic DB adapter.

Deploy the Adapter

1. In BSM, access the Package Manager. For details, see "Package Manager Page" in the *RTSM Administration Guide*.
2. Click the **Deploy Packages to Server (from local disk)** icon  and browse to your adapter package. Select the package and click **Open**, then click **Deploy** to display the package in the Package Manager.
3. Select your package in the list and click the **View package resources** icon  to verify that the package contents are recognized by Package Manager.

Edit the Adapter

Once you have created and deployed the adapter, you can then edit it within BSM. For details, see "Adapter Management" in the *Data Flow Management Guide*.

Create an Integration Point

In this step you check that the federation is working. That is, that the connection is valid and that the XML file is valid. However, this check does not verify that the XML is mapping to the correct fields in the RDBMS.

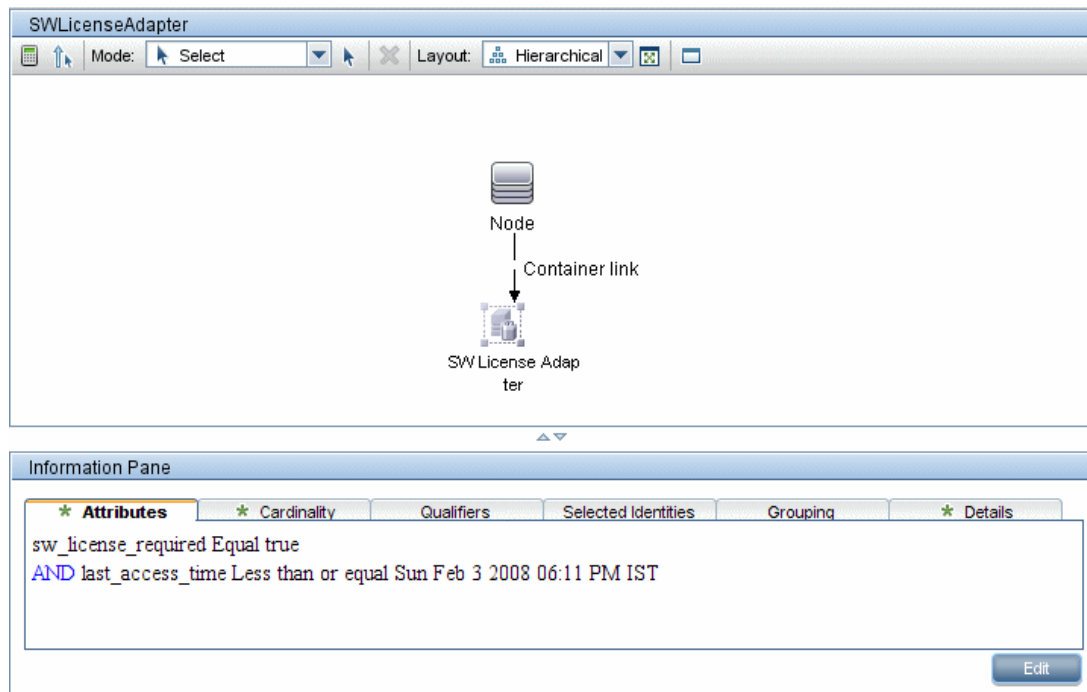
1. In BSM, access the Integration Studio (**Admin > RTSM Administration > Data Flow Management > Integration Studio**).
2. Create an integration point. For details, see New Integration Point/Edit Integration Point Dialog Box in the *Data Flow Management Guide*.

The Federation tab displays all CITs that can be federated using this integration point. For details, see Federation Tab in the *Data Flow Management Guide*.

Create a View


In this step you create a view that enables you to view instances of the CIT.

1. In BSM, access the Modeling Studio (**Admin > RTSM Administration > Modeling > Modeling Studio**).
2. Create a view. For details, see Create a Pattern View in the Modeling Guide.
3. You can add conditions to the TQL, for example, the last access time is greater than six months:



Calculate the Results

In this step you check the results.

1. In BSM, access the Modeling Studio (**Admin > RTSM Administration > Modeling > Modeling Studio**).
2. Open a view.
3. Calculate results by clicking the **Calculate Query Result Count** button .
4. Click the **Preview** button to view the CIs in the view.

View the Results

In this step you view the results and debug problems in the procedure. For example, if nothing is shown in the view, check the definitions in the **orm.xml** file; remove the relationship attributes and reload the adapter.

1. In BSM, access the IT Universe Manager (**Admin > RTSM Administration > Modeling > IT Universe Manager**).
2. Select a CI. The Properties tab displays the results of the federation.

View Reports

In this step you view Topology reports. For details, see Topology Reports Overview in the Modeling Guide.

Enable Log Files

To understand the calculation flows, adapter lifecycle, and to view debug information, you can consult the log files. For details, see ["Adapter Log Files" on page 172](#).

Use Eclipse to Map Between CIT Attributes and Database Tables

Caution: This procedure is intended for users with an advanced knowledge of content development. For any questions, contact HP Software Support.

This task describes how to install and use the JPA plugin, provided with the J2EE edition of Eclipse, to:

- Enable graphical mapping between RTSM class attributes and database table columns.
- Enable manual editing of the mapping file (`orm.xml`), while providing correctness. The correctness check includes a syntax check as well as verification that the class attributes and mapped database table columns are stated correctly.
- Enable deployment of the mapping file to the RTSM server and to view the errors, as a further correctness check.
- Define a sample query on the RTSM server and run it directly from Eclipse, to test the mapping file.

Version 1.1 of the plugin is compatible with UCMDB version 9.01 or later and Eclipse IDE for Java EE Developers, version 1.2.2.20100217-2310 or later.

This task includes the following steps:

1. Prerequisites

Install the latest update for **Java Runtime Environment (JRE) 6** on the machine where you will run Eclipse from the following site:

<http://java.sun.com/javase/downloads/index.jsp>.

2. Installation

- a. Download and extract **Eclipse IDE for Java EE Developers** from <http://www.eclipse.org/downloads> to a local folder, for example, **C:\Program Files\ eclipse**.
- b. Copy **com.hp.plugin.import_cmdb_model_1.0.jar** from **C:\hp\UCMDB\UCMDBServer\tools\db-adapter-eclipse-plugin\bin** to **C:\Program Files\Eclipse\plugins**.
- c. Launch **C:\Program Files\Eclipse\eclipse.exe**. If a message is displayed that the Java virtual machine is not found, launch **eclipse.exe** with the following command line:

```
"C:\Program Files\eclipse\eclipse.exe" -vm "<JRE installation folder>\bin"
```

3. Prepare the Work Environment

In this step, you set up the workspace, database, connections, and driver properties.

- a. Extract the file **workspaces_gdb.zip** from **C:\hp\UCMDB\UCMDBServer\tools\db-adapter-eclipse-plugin\workspace** into **C:\Documents and**

Settings\All Users.

Note: You must use the exact folder path. If you unzip the file to the wrong path or leave the file unzipped, the procedure will not work.

- b. In Eclipse, choose **File > Switch Workspace > Other:**
If you are working with:
 - o SQL Server, select the following folder: **C:\Documents and Settings\All Users\workspace_gdb_sqlserver.**
 - o MySQL, select the following folder: **C:\Documents and Settings\All Users\workspace_gdb_mysql.**
 - o Oracle, select the following folder: **C:\Documents and Settings\All Users\workspace_gdb_oracle.**
- c. Click **OK.**
- d. In Eclipse, display the Project Explorer view and select **<Active project> > JPA Content > persistence.xml > <active project name> > orm.xml.**
- e. In the Data Source Explorer view (the bottom left pane), right-click the database connection and select the **Properties** menu.
- f. In the **Properties for <Connection name>** dialog box, select **Common** and select the **Connect every time the workbench is started** check box. Select **Driver Properties** and fill in the connection properties. Click **Test Connection** and verify that the connection is working. Click **OK.**
- g. In the Data Source Explorer view, right-click the database connection and click **Connect.** A tree containing the database schemas and tables is displayed under the database connection icon.

4. Create an Adapter

Create an adapter using the guidelines in ["Step 1: Create an Adapter" on page 31.](#)

5. Configure the RTSM Plugin

- a. In Eclipse, click **UCMDB > Settings** to open the **CMDB Settings** dialog box.
- b. If not already selected, select the newly created JPA project as the Active project.
- c. Enter the RTSM host name, for example, **localhost** or **labm1.itdep1**. There is no need to include the port number or **http://** prefix in the address.
- d. Fill in the user name and password for accessing the RTSM API, usually **admin/admin.**
- e. Make sure that the **C:\hp** folder on the RTSM server is mapped as a network drive.
- f. Select the base folder of the relevant adapter under **C:\hp**. The base folder is the one that contains the **dbAdapter.jar** file and the **META-INF** subfolder. Its path should be **<HP BSM root directory>\odb\runtime\fcmdb\CodeBase\<adapter name>**. Verify that there is no backslash (\) at the end.

6. Import the BSM Class Model

In this step, you select the CITs to be mapped as JPA entities.

- a. Click **UCMDB > Import CMDB Class Model** to open the **CI Types Selection** dialog box.
- b. Select the CI types that you intend to map as JPA entities. Click **OK**. The CI types are imported as Java classes. Verify that they appear under the **src** folder of the active project.

7. Build the ORM File – Map BSM Classes to Database Tables

In this step, you map the Java classes (that you imported in the previous step) to the database tables.

- a. Make sure the DB connection is connected. Right-click the active project (called myProject by default) in Project Explorer. Select the JPA view, select the **Override default schema from connection** check box, and select the relevant database schema. Click **OK**.
- b. Map a CIT: In the JPA Structure view, right-click the **Entity Mappings** branch and select **Add Class**. The **Add Persistent Class** dialog box opens. Do not change the **Map as** field (**Entity**).
- c. Click **Browse** and select the BSM class to be mapped (all BSM classes belong to the **generic_db_adapter** package).
- d. Click **OK** in both dialog boxes. The selected class is displayed under the **Entity Mappings** branch in the JPA Structure view.

Note: If the entity appears without an attribute tree, right-click the active project in the Project Explorer view. Choose **Close** and then **Open**.

- e. In the JPA Details view, select the primary database table to which the BSM class should be mapped. Leave all other fields unchanged.

8. Map IDs

According to JPA standards, each persistent class must have at least one ID attribute. For BSM classes, you can map up to three attributes as IDs. Potential ID attributes are called **id1**, **id2**, and **id3**. To map an ID attribute:

- a. Expand the corresponding class under the **Entity Mappings** branch in the JPA Structure view, right-click the relevant attribute (for example, **id1**), and select **Add Attribute to XML and Map....**
- b. The **Add Persistent Attribute** dialog box opens. Select **Id** in the **Map as** field and click **OK**.
- c. In the JPA Details view, select the database table column to which the ID field should be mapped.

9. Map Attributes

In this step, you map attributes to the database columns.

- a. Expand the corresponding class under the **Entity Mappings** branch in the JPA Structure view, right-click the relevant attribute (for example, **host_hostname**), and select **Add Attribute to XML and Map...**
- b. The **Add Persistent Attribute** dialog box opens. Select **Basic** in the **Map as** field and click **OK**.
- c. In the JPA Details view, select the database table column to which the attribute field should be mapped.

10. Map a Valid Link

Perform the steps described above in the step "[Build the ORM File – Map BSM Classes to Database Tables](#)" on the previous page for mapping a BSM class denoting a valid link. The name of each such class takes the following structure: **<end1 entity name>_<link name>_<end 2 entity name>**. For example, a **Contains** link between a host and a location is denoted by a Java class whose name is **generic_db_adapter.host_contains_location**. For details, see "[The reconciliation_rules.txt File \(for backwards compatibility\)](#)" on page 152.

- a. Map the ID attributes of the link class as described in "[Map IDs](#)" on the previous page. For each ID attribute, expand the **Details** check box group in the JPA Details view and clear the **Insertable** and **Updateable** check boxes.
- b. Map the **end1** and **end2** attributes of the link class as follows: For each of the **end1** and **end2** attributes of the link class:
 - Expand the corresponding class under the **Entity Mappings** branch in the JPA Structure view, right-click the relevant attribute (for example, **end1**), and select **Add Attribute to XML and Map...**
 - In the **Add Persistent Attribute** dialog box, select **Many to One** or **One to One** in the **Map as** field.
 - Select **Many to One** if the specified **end1** or **end2** CI can have multiple links of this type. Otherwise, select **One to One**. For example, for a **host_contains_ip** link the **host** end should be mapped as **Many to One**, since one host can have multiple IPs, and the **ip** end should be mapped as **One to One**, since one IP can have only a single host.
 - In the JPA Details view, select **Target entity**, for example, **generic_db_adapter.host**.
 - In the **Join Columns** section of the JPA Details view, check **Override Default**. Click **Edit**. In the **Edit Join Column** dialog box, select the foreign key column of the link database table that points to an entry in the **end1/end2** target entity's table. If the referenced column name in the **end1/end2** target entity's table is mapped to its ID attribute, leave the **Referenced Column Name** unchanged. Otherwise, select the name of the column to which the foreign key column points. Clear the **Insertable** and **Updateable** check boxes and click **OK**.
 - If the **end1/end2** target entity has more than one ID, click the **Add** button to add additional join columns and map them in the same way as described in the previous step.

11. Build the ORM File – Use Secondary Tables

JPA enables a Java class to be mapped to more than one database table. For example, **Host** can be mapped to the **Device** table to enable persistence of most of its attributes and to the

NetworkNames table to enable persistence of **host_hostName**. In this case, **Device** is the primary table and **NetworkNames** is the secondary table. Any number of secondary tables can be defined. The only condition is that there must be a one-to-one relationship between the entries of the primary and secondary tables.

12. Define a Secondary Table

Select the appropriate class in the JPA Structure view. In the **JPA Details** view, access the **Secondary Tables** section and click **Add**. In the **Add Secondary Table** dialog box, select the appropriate secondary table. Leave the other fields unchanged.

If the primary and the secondary table do not have the same primary keys, configure the join columns in the **Primary Key Join Columns** section of the **JPA Details** view.

13. Map an Attribute to a Secondary Table

You map a class attribute to a field of a secondary table as follows:

- a. Map the attribute as described above in ["Map Attributes" on page 136](#).
- b. In the **Column** section of the JPA Details view, select the secondary table name in the **Table** field, to replace the default value.

14. Use an Existing ORM File as a Base

To use an existing **orm.xml** file as a basis for the one you are developing, perform the following steps:

- a. Verify that all CITs mapped in the existing **orm.xml** file are imported into the active Eclipse project.
- b. Select and copy all or part of the entity mappings from the existing file.
- c. Select the **Source** tab of the **orm.xml** file in the Eclipse JPA perspective.
- d. Paste all copied entity mappings under the **<entity-mappings>** tag of the edited **orm.xml** file, beneath the **<schema>** tag. Make sure that the schema tag is configured as described above in the step ["Build the ORM File – Map BSM Classes to Database Tables" on page 136](#). All pasted entities now appear in the JPA Structure view. From now on, mappings can be edited both graphically and manually through the xml code of the **orm.xml** file.
- e. Click **Save**.

15. Importing an Existing ORM File from an Adapter

If an adapter already exists, the Eclipse Plugin can be used to edit its ORM file graphically. Import the **orm.xml** file into Eclipse, edit it using the plugin and then deploy it back to the BSM machine. To import the ORM file, press the button on the Eclipse toolbar. A confirmation dialog is displayed. Click **OK**. The ORM file is copied from the BSM machine to the active Eclipse project and all relevant classes are imported from the BSM class model.

If the relevant classes do not appear in the JPA Structure view, right-click the active project in the Project Explorer view, choose **Close** and then **Open**.

From now on, the ORM file can be edited graphically using Eclipse, and then deployed back to the BSM machine as described below in ["Deploy the ORM File to the RTSM" on the next page](#).

16. Check the Correctness of the orm.xml File – Built-in Correctness

Check

The Eclipse JPA plugin checks if any errors are present and marks them in the **orm.xml** file. Both syntax (for example, wrong tag name, unclosed tag, missing ID) and mapping errors (for example, wrong attribute name or database table field name) are checked. If there are errors, their description appears in the **Problems** view.

17. Create a New Integration Point

If no integration point exists in the RTSM for this adapter, you can create it in the Integration Studio. For details, see Integration Studio in the *Data Flow Management Guide*.

Fill in the integration point name in the dialog box that opens. The **orm.xml** file is copied to the adapter folder. An integration point is created with all the imported CI types as its supported classes, except for multinode CITs, if they are configured in the **reconciliation_rules.txt** file. For details, see "[The reconciliation_rules.txt File \(for backwards compatibility\)](#)" on page 152.

18. Deploy the ORM File to the RTSM

Save the **orm.xml** file and deploy it to the BSM server by clicking **UCMDB > Deploy ORM**. The **orm.xml** file is copied to the adapter folder and the adapter is reloaded. The operation result is shown in an **Operation Result** dialog box. If any error occurs during the reload process, the Java exception stack trace is displayed in the dialog box. If no integration point has yet been defined using the adapter, no mapping errors are detected upon deployment.

19. Run a Sample TQL Query

- a. Define a query (not a view) in the Modeling Studio. For details, see Modeling Studio in the *Data Flow Management Guide*.
- b. Create an integration point using the adapter that you created in the step "[Create a New Integration Point](#)" above. For details, see New Integration Point/Edit Integration Point Dialog Box in the *Data Flow Management Guide*.
- c. During the creation of the adapter, verify that the CI types that should participate in the query are supported by this integration point.
- d. When configuring the RTSM plugin, use this sample query name in the Settings dialog box. For details, see step 5 above.
- e. Click the **Run TWL** button to run a sample TQL and verify whether it returns the required results using the newly created **orm.xml** file.

Adapter Configuration Files

The files discussed in this section are located in the **db-adapter.zip** package in the **<HP BSM root directory>\odb\content\adapters** folder.

This section includes the following topics:

- ["The adapter.conf File" on the next page](#)
- ["The simplifiedConfiguration.xml File" on page 142](#)
- ["The orm.xml File" on page 143](#)
- ["The reconciliation_types.txt file" on page 152](#)
- ["The reconciliation_rules.txt File \(for backwards compatibility\)" on page 152](#)
- ["The transformations.txt File" on page 154](#)
- ["The discriminator.properties File" on page 155](#)
- ["The replication_config.txt File" on page 156](#)
- ["The fixed_values.txt File" on page 156](#)
- ["The persistence.xml File" on page 156](#)

General Configuration

- **adapter.conf.** The adapter configuration file. For details, see ["The adapter.conf File" on the next page](#).

Simple Configuration

- **simplifiedConfiguration.xml.** Configuration file that replaces **orm.xml**, **transformations.txt**, and **reconciliation_rules.txt** with less capabilities. For details, see ["The simplifiedConfiguration.xml File" on page 142](#).

Advanced Configuration

- **orm.xml.** The object-relational mapping file in which you map between RTSM CITs and database tables. For details, see ["The orm.xml File" on page 143](#).
- **reconciliation_types.txt.** Contains the rules that are used to configure the reconciliation types. For details, see ["The reconciliation_types.txt file" on page 152](#).
- **reconciliation_rules.txt.** Contains the reconciliation rules. For details, see ["The reconciliation_rules.txt File \(for backwards compatibility\)" on page 152](#).
- **transformations.txt.** Transformations file in which you specify the converters to apply to convert from the RTSM value to the database value, and vice versa. For details, see ["The transformations.txt File" on page 154](#).
- **Discriminator.properties.** This file maps each supported CI type to a comma-separated list of possible corresponding values. For details, see ["The discriminator.properties File" on page 155](#).
- **Replication_config.txt.** This file contains a comma-separated list of CI and relationship types whose property conditions are supported by the replication plugin. For details, see ["The replication_config.txt File" on page 156](#).

- **Fixed_values.txt.** This file enables you to configure fixed values for specific attributes of certain CITs. For details, see ["The fixed_values.txt File" on page 156](#).

Hibernate Configuration

- **persistence.xml.** Used to override out-of-the-box Hibernate configurations. For details, see ["The persistence.xml File" on page 156](#).

The adapter.conf File

This file contains the following settings:

- **use.simplified.xml.config=false.true:** uses simplifiedConfiguration.xml.

Note: Using this file means that `orm.xml`, `transformations.txt`, and `reconciliation_rules.txt` are replaced with fewer capabilities.

- **dal.ids.chunk.size=300.** Do not change this value.
- **dal.use.persistence.xml=false.true:** the adapter reads the Hibernate configuration from `persistence.xml`.

Note: It is not recommended to override the Hibernate configuration.

- **performance.memory.id.filtering=true.** When the GDBA executes TQLS, in some cases a large number of IDs may be retrieved and sent back to the database using SQL. To avoid this excessive work and improve performance, the GDBA attempts to read the entire view/table and filters the results in-memory.
- **id.reconciliation.cmdb.id.type=string/bytes.** When mapping the Generic DB adapter using ID Reconciliation (for information, see the step ["Configure the reconciliation_types.txt File \(for the BSM 9.x default mapping engine\)"](#) in ["Implement the Mapping Engine" on page 206](#), you can either map the `cmdb_id` to a **string** or **bytes/raw** column type by changing the **META-INF/adapter.conf** property.
- **performance.enable.single.sql=true.** This is an optional parameter. If it does not appear in the file, its default value is **true**. When **true**, the Generic Database Adapter tries to generate a single SQL statement for each query being executed (either for population or a federated query). Using a single SQL statement improves the performance and memory consumption of the Generic Database Adapter. When **false**, the Generic Database Adapter generates multiple SQL statements, which may take longer and consume more memory than a single one. Even when this attribute is set to **true**, the adapter does not generate a single SQL statement in the following scenarios:
 - The database the adapter connects to is not on an Oracle or SQL Server.
 - The TQL being executed contains a cardinality condition other than `0..*` and `1..*` (for example, if there is a cardinality condition like `2..*` or `0..2`).

The simplifiedConfiguration.xml File

This file is used for simple mapping of BSM classes to database tables. To access the template for editing the file, navigate to **Admin > RTSM Administration > Adapter Management > db-adapter > Configuration files**.

This section includes the following topics:

- ["The simplifiedConfiguration.xml File Template" below](#)
- ["Limitations" on the next page](#)

The simplifiedConfiguration.xml File Template

The **CMDB-class-name** property is the multinode type (the node to which federated CITs connect in the TQL):

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=" ../META-
CONF/simplifiedConfiguration.xsd">
    <CMDB-class CMDB-class-name="node" default-table-name="[table_
name]">
        <primary-key column-name="[column_name]" />
```

reconciliation-by-two-nodes. Reconciliation can be done using one node or two nodes. In this case example, reconciliation uses two nodes.

connected-node-CMDB-class-name. The second class type needed in the reconciliation TQL.

CMDB-link-type. The relationship type needed in the reconciliation TQL.

link-direction. The direction of the relationship in the reconciliation TQL (from `node` to `ip_address` or from `ip_address` to `node`):

```
    <reconciliation-by-two-nodes connected-node-CMDB-class-
name="ip_address" CMDB-link-type="containment" link-direction="main-
to-connected">
```

The reconciliation expression is in the form of ORs and each OR includes ANDs.

is-ordered. Determines if reconciliation is done in order form or by a regular OR comparison.

```
    <or is-ordered="true">
```

If the reconciliation property is retrieved from the main class (the multinode), use the **attribute** tag, otherwise use the **connected-node-attribute** tag.

ignore-case.true: when data in the BSM class model is compared with data in the RDBMS, case does not matter:

```
        <attribute CMDB-attribute-name="name" column-name="
[column_name]" ignore-case="true" />
```

The column name is the name of the foreign key column (the column with values that point to the multinode primary key column).

If the multinode primary key column is composed of several columns, there needs to be several foreign key columns, one for each primary key column.

```
<foreign-primary-key column-name="[column_name]" CMDB-class-  
primary-key-column="[column_name]" />
```

If there are few primary key columns, duplicate this column.

```
<primary-key column-name="[column_name]" />
```

The **from-CMDB-converter** and **to-CMDB-converter** properties are Java classes that implement the following interfaces:

- com.mercury.topaz.fcmbd.adapters.dbAdapter.dal.transform.FcmbdDalTransformerFromExternalDB
- com.mercury.topaz.fcmbd.adapters.dbAdapter.dal.transform.FcmbdDalTransformerToExternalDB

Use these converters if the value in the RTSM and in the database are not the same.

In this example, `GenericEnumTransformer` is used to convert the enumerator according to the XML file that is written inside the parenthesis (**generic-enum-transformer-example.xml**):

```
<attribute CMDB-attribute-name="[CMDB_attribute_name]" column-  
name="[column_name]" from-CMDB-  
con-  
verter="com.mercury.topaz.fcmbd.adapters.dbAdapter.dal.transform.impl.  
GenericEnumTransformer(generic-enum-transformer-example.xml)" to-CMDB-  
con-  
verter="com.mercury.topaz.fcmbd.adapters.dbAdapter.dal.transform.impl.  
GenericEnumTransformer(generic-enum-transformer-example.xml)" />  
<attribute CMDB-attribute-name="[CMDB_attribute_name]" column-  
name="[column_name]" />  
<attribute CMDB-attribute-name="[CMDB_attribute_name]" column-  
name="[column_name]" />  
</class>  
</generic-DB-adapter-config>
```

Limitations

- Can be used to map only TQL queries containing one node (in the database source). For example, you can run a `node > ticket` and a `ticket` TQL query. To bring the hierarchy of nodes from the database, you must use the advanced **orm.xml** file.
- Only one-to-many relations are supported. For example, you can bring one or more tickets on each node. You cannot bring tickets that belong to more than one node.
- You cannot connect the same class to different types of RTSM CITs. For example, if you define that `ticket` is connected to `node`, it cannot be connected to `application` as well.

The orm.xml File

This file is used for mapping RTSM CITs to database tables.

A template to use for creating a new file is located in the
<HP BSM root directory>\odb\runtime\fcmdb\CodeBase\GenericDBAdapter\META-INF
directory.

To edit the XML file for a deployed adapter, navigate to **Admin > RTSM Administration > Adapter Management > db-adapter > Configuration files**.

This section includes the following topics:

- ["The orm.xml File Template" below](#)
- ["Multiple ORM files" on page 147](#)
- ["Naming Conventions" on page 147](#)
- ["The orm.xml File" on the previous page](#)
- ["The orm.xml Schema" on page 147](#)

The orm.xml File Template

```
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.0"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm
http://java.sun.com/xml/ns/persistence/orm_1_0.xsd">
  <description>Generic DB adapter orm</description>
```

Do not change the package name.

```
<package>generic_db_adapter</package>
```

entity. The RTSM CIT name. This is the multinode entity.

Make sure that **class** includes a **generic_db_adapter.** prefix.

```
<entity class="generic_db_adapter.node">
  <table name="[table_name]" />
```

Use a secondary table if the entity is mapped to more than one table.

```
<secondary-table name="" />
<attributes>
```

For a single table inheritance with discriminator, use the following code:

```
<inheritance strategy="SINGLE_TABLE" />
<discriminator-value>node</discriminator-value>
<discriminator-column name="[column_name]" />
```

Attributes with tag **id** are the primary key columns. Make sure that the naming convention for these primary key columns are **idX** (id1, id2, and so on) where **X** is the column index in the primary key.

```
<id name="id1">
```

Change only the column name of the primary key.

```
<column updatable="false" insertable="false" name="
[column_name]" />
```



```
        <generated-value strategy="TABLE" />
    </id>
```

basic. Used to declare the RTSM attributes. Make sure to edit only **name** and **column_name** properties.

```
        <basic name="name">
            <column updatable="false" insertable="false" name="
[column_name]" />
        </basic>
```

For a single table inheritance with discriminator, map the extending classes as follows:

```
    <entity name="[cmdb_class_name]" class="generic_db_adapter.nt"
name="nt">

        <discriminator-value>nt</discriminator-value>
        <attributes>
    </entity>
    <entity class="generic_db_adapter.unix" name="unix">
        <discriminator-value>unix</discriminator-value>
        <attributes>
    </entity>
    <entity name="[CMDB_class_name]" class="generic_db_adapter.[CMDB
[cmdb_class_name]">
        <table name="[default_table_name]" />
        <secondary-table name="" />
        <attributes>
            <id name="id1">
                <column updatable="false" insertable="false" name="
[column_name]" />
                <generated-value strategy="TABLE" />
            </id>
            <id name="id2">
                <column updatable="false" insertable="false" name="
[column_name]" />
                <generated-value strategy="TABLE" />
            </id>
            <id name="id3">
                <column updatable="false" insertable="false" name="
[column_name]" />
                <generated-value strategy="TABLE" />
            </id>
```

The following example shows a RTSM attribute name with no prefix:

```
        <basic name="[CMDB_attribute_name]">
            <column updatable="false" insertable="false" name="
[column_name]" />
        </basic>
        <basic name="[CMDB_attribute_name]">
            <column updatable="false" insertable="false" name="
[column_name]" />
```

```
        </basic>
        <basic name="[CMDB_attribute_name]">
            <column updatable="false" insertable="false" name="
[column_name]" />
        </basic>
    </attributes>
</entity>
```

This is a relationship entity. The naming convention is **end1Type_linkType_end2Type**. In this example **end1Type** is **node** and the **linkType** is **composition**.

```
<entity name="node_composition_[CMDB_class_name]" class="generic_
db_adapter.node_composition_[CMDB_class_name]">
    <table name="[default_table_name]" />
    <attributes>
        <id name="id1">
            <column updatable="false" insertable="false" name="
[column_name]" />
            <generated-value strategy="TABLE" />
        </id>
```

The target entity is the entity that this property is pointing to. In this example, **end1** is mapped to **node** entity.

many-to-one. Many relationships can be connected to one node.

join-column. The column that contains **end1** IDs (the target entity IDs).

referenced-column-name. The column name in the target entity (**node**) that contain the IDs that are used in the join column.

```
        <many-to-one target-entity="node" name="end1">
            <join-column updatable="false" insertable="false"
referenced-column-name="[column_name]" name="[column_name]" />
        </many-to-one>
```

one-to-one. One relationship can be connected to one **[CMDB_class_name]**.

```
        <one-to-one target-entity="[CMDB_class_name]" name="end2">
            <join-column updatable="false" insertable="false"
referenced-column-name="" name="[column_name]" />
        </one-to-one>
    </attributes>
</entity>
</entity-mappings>
```

node attribute. This is an example of how to add a node attribute.

```
<entity class="generic_db_adapter.host_node">
    <discriminator-value>host_node</discriminator-value>
    <attributes/>
</entity>

<entity class="generic_db_adapter.nt">
```

```
<discriminator-value>nt</discriminator-value>

<attributes>

<basic name="nt_servicepack">

<column updatable="false" insertable="false" name="specific_type_
value"/>

</basic>

</attributes>

</entity>
```

Multiple ORM files

Multiple mapping files are supported. Each mapping file name should end with **orm.xml**. All mapping files should be placed under the META-INF folder of the adapter.

Naming Conventions

- In each entity, the class property must match the name property with the prefix of `generic_db_adapter`.
- Primary key columns must take names of the form **idX** where **X = 1, 2, ...**, according to the number of primary keys in the table.
- Attribute names must match class attribute names even as regards case.
- The relationship name takes the form `end1Type_linkType_end2Type`.
- RTSM CITs, which are also reserved words in Java, should be prefixed by **gdba_**. For example, for the RTSM CIT **goto**, the ORM entity should be named **gdba_goto**.

Using Inline SQL Statements Instead of Table Names

You can map entities to inline `select` clauses instead of to database tables. This is equivalent to defining a view in the database and mapping an entity to this view. For example:

```
<entity class="generic_db_adapter.node">
  <table name="(select d.id as id1, d.name as name , d.os as
host_os from
Device d)" />
```

In this example, the node attributes should be mapped to columns `id1`, `name`, and `host_os`, rather than `id`, `name`, and `os`.

The following limitations apply:

- The inline SQL statement is available only when using Hibernate as the JPA provider.
- Round brackets around the inline SQL select clause are mandatory.
- The **<schema>** element should not be present in the **orm.xml** file. In the case of Microsoft SQL Server 2005, this means that all table names should be prefixed with `dbo.`, rather than defining them globally by `<schema>dbo</schema>`.

The orm.xml Schema

The following table explains the common elements of the **orm.xml** file. The complete schema can

be found at http://java.sun.com/xml/ns/persistence/orm_1_0.xsd. The list is not complete, and it mainly explains the specific behavior of the standard Java Persistence API for the Generic Database Adapter.

Element Name and Path	Description	Attributes
entity-mappings	The root element for the entity mapping document. This element should be exactly the same as the one given in the GDBA sample files.	
description (entity-mappings)	A free text description of the entity mapping document. (Optional)	
package (entity-mappings)	The name of the Java package that will contain the mapping classes. Should always contain the text <code>generic_db_adapter</code> .	<ol style="list-style-type: none"> Name: name Description: The name of the UCMDB CI type to which this entity is mapped. If this entity is mapped to a link in the CMDB, the name of the entity should be in the format <code><end_1>_<link_name>_<end_2></code>. For example, <code>node_composition_cpu</code> defines an entity that will be mapped to the composition link between a node and a CPU. If the name of the CI type is the same as the name of the Java class without the package prefix, this field can be omitted. Is required?: Optional Type: String Name: class Description: The fully qualified name of the Java class that will be created for this DB entity. The name of the Java class' package should be the same as the name given in the <code>package</code> element. You may not use Java reserved words, such as <code>interface</code> or <code>switch</code>, as the class name. Instead, add the prefix <code>gdba_</code> to the name (so <code>interface</code> will be <code>generic_db_adapter.gdba_interface</code>). Is required?: Required Type: String

Element Name and Path	Description	Attributes
table (entity-mappings>entity)	This element defines the primary table of the DB entity. Can only appear once. Required.	Name: name Description: The name of the primary table. If the name of the table does not contain the schema to which it belongs, the table will be searched only in the schema of the user that was used to create the integration point. This can also be any a valid SELECT statement. If this is a SELECT statement, it must be encapsulated with parentheses. Is required?: Required Type: String
secondary-table (entity-mappings > entity)	This element may be used to define a secondary table for the DB entity. This table must be connected to the primary table with a 1-to-1 relationship. You may define more than one secondary table. Optional.	Name: name Description: The name of the secondary table. If the name of the table does not contain the schema to which it belongs, the table will be searched only in the schema of the user that was used to create the integration point. This can also be any a valid SELECT statement. If this is a SELECT statement, it must be encapsulated with parentheses. Is required?: Required Type: String
primary-key-join-column (entity-mappings > entity > secondary-table)	If the secondary table and primary table are not connected using fields with the same name, this element defines the name of the primary key field in the secondary table that needs to be connected to the primary key field of the primary table.	Name: name Description: The name of the primary key field in the secondary table. If this element does not exist, it is assumed that the primary key field has the same name as the primary key field of the primary table. Is required?: Optional Type: String

Element Name and Path	Description	Attributes
inheritance (entity-mappings > entity)	If the current entity is the parent entity for a family of DB entities, then use this element to mark it as such. Optional.	Name: strategy Description: Defines the way the inheritance is implemented in your DB. Is required?: Required Type: One of the following values: <ul style="list-style-type: none"> SINGLE_TABLE: This entity and all child entities exist in the same table. JOINED: The child entities are in joined tables. TABLE_PER_CLASS: Each entity is completely defined by a separate table.
discriminator-column (entity-mappings > entity)	If the inheritance is of type SINGLE_TABLE, this element is used to define the name of the field used to determine the type of entity for each row.	Name: name Description: The name of the discriminator column. Is required?: Required Type: String
discriminator-value (entity-mappings > entity)	This element defines the type of the specific entity in the inheritance tree. This name needs to be the same as the name defined in the discriminator.properties file for the value group of this specific entity type.	
attributes (entity-mappings > entity)	The root element for all of the attribute mappings for an entity.	
id (entity-mappings > entity attributes)	This element defines the key field for the entity. There must be at least one id field defined. If more than one id element exists, its fields create a compound key for the entity. You should try and avoid compound keys for CI entities (not for links).	Name: name Description: A string of type idX, where X is a number between 1 and 9. The first id should be marked as id1, the second as id2 and so on. This is NOT the name of the key attribute in UCMDDB. Is required?: Required Type: String

Element Name and Path	Description	Attributes
basic (entity-mappings > entity attributes)	This element defines a mapping between a field in the table, which is not part of the table's primary key, and a UCMDB attribute.	Name: name Description: The name of the UCMDB attribute to which the field is mapped. This attribute must exist in the UCMDB CI type to which the current entity is mapped. Is required?: Required Type: String
column (entity-mappings > entity > attributes > id -OR- (entity-mappings > entity > attributes > basic)	Defines the name of the column in the table for basic mapping or an id field.	<ol style="list-style-type: none"> 1. Name: name Description: The name of the field. Is required?: Required Type: String 2. Name: table Description: The name of the table to which the field belongs. This must be either the primary table or one of the secondary tables defined for the entity. If this attribute is omitted, it is assumed that the field belongs to the primary table. Is required: Optional Type: String
one-to-one (entity-mappings > entity > attributes)	Defines a column whose value is in another table, and the two tables are connected using a one-to-one relationship. This element is only supported for link entity mappings and not for other CI types. This is the only way to define a mapping between a table and a UCMDB link.	<ol style="list-style-type: none"> 1. Name: name Description: Which of the two ends this field represents. Is required?: Required Type: Either <code>end1</code> or <code>end2</code> 2. Name: target-entity Description: The name of the entity to which the end refers. Is required?: Required Type: One of the entity names defined in the entity mapping document.

Element Name and Path	Description	Attributes
join-column (entity-mappings > entity attributes > one-to-one)	Defines the way to join the target-entity defined in the parent one-to-one element and the current entity.	<ol style="list-style-type: none">Name: name Description: The name of the field in the current table that will be used to perform the one-to-one join. Is required?: Required Type: StringName: name Description: The name of a field in the joint entity by which to perform the join. If this attribute is omitted, it is assumed that the joint table has a column with the same name as the field defined in the name attribute. Is required?: Optional Type: String

The reconciliation_types.txt file

This file is used to configure the reconciliation types.

Each row in the file represents a RTSM CIT that is connected to a federated database CIT in the TQL query.

The reconciliation_rules.txt File (for backwards compatibility)

This file is used to configure the reconciliation rules if you want to perform reconciliation when the DBMappingEngine is configured in the adapter. If you do not use the DBMappingEngine, the generic RTSM reconciliation mechanism is used and there is no need to configure this file.

Each row in the file represents a rule. For example:

```
multinode[node] expression[^node.name OR ip_address.name] end1_type  
[node]  
end2_type[ip_address] link_type[containment]
```

The multinode is filled with the multinode name (the RTSM CIT that is connected to the federated database CIT in the TQL query).

This expression includes the logic that decides whether two multinodes are equal (one multinode in the RTSM and the other in the database source).

The expression is composed of ORs or ANDs.

The convention regarding attribute names in the expression part is `[className].[attributeName]`. For example, `attributeName` in the `ip_address` class is written `ip_address.name`.

For an ordered match (if the first `OR` sub-expression returns an answer that the multinode is not equal, the second `OR` sub-expression is not compared), then use `ordered expression` instead of `expression`.

To ignore case during a comparison, use the control (^) sign.

The parameters `end1_type`, `end2_type` and `link_type` are used only if the reconciliation TQL query contains two nodes and not just a multinode. In this case, the reconciliation TQL query is `end1_type > (link_type) > end2_type`.

There is no need to add the relevant layout as it is taken from the expression.

Types of Reconciliation Rules

Reconciliation rules take the form of OR and AND conditions. You can define these rules on several different nodes (for example, node is identified by `name from node AND/OR name from ip_address`).

The following options find a match:

- **Ordered match.** The reconciliation expression is read from left to right. Two `OR` sub-expressions are considered equal if they have values and they are equal. Two `OR` sub-expressions are considered not equal if both have values and they are not equal. For any other case there is no decision, and the next `OR` sub-expression is tested for equality.

name from node OR from ip_address. If both the RTSM and the data source include `name` and they are equal, the nodes are considered as equal. If both have `name` but they are not equal, the nodes are considered not equal without testing the `name of ip_address`. If either the RTSM or the data source is missing `name of node`, the `name of ip_address` is checked.

- **Regular match.** If there is equality in one of the `OR` sub-expressions, the RTSM and the data source are considered equal.

name from node OR from ip_address. If there is no match on `name of node`, `name of ip_address` is checked for equality.

For complex reconciliations, where the reconciliation entity is modeled in the class model as several CITs with relationships (such as `node`), the mapping of a superset node includes all relevant attributes from all modeled CITs.

Note: As a result, there is a limitation that all reconciliation attributes in the data source should reside in tables that share the same primary key.

Another limitation states that the reconciliation TQL query should have no more than two nodes. For example, the `node > ticket` TQL query has a node in the RTSM and a ticket in the data source.

To reconcile the results, `name` must be retrieved from the node and/or `ip_address`.

If the `name` in the RTSM is in the format of `*.m.com`, a converter can be used from RTSM to the federated database, and vice versa, to convert these values.

The `node_id` column in the database ticket table is used to connect between the entities (the defined association can also be made in a node table):

DB Node		DB IP_Address	
PK	node_id	PK	ip_id
	name		name

DB Ticket	
PK	ticket_id
	node_id

Note: The three tables must be part of the federated RDBMS source and not the RTSM database.

The transformations.txt File

This file contains all the converter definitions.

The format is that each line contains a new definition.

The transformations.txt File Template

```
entity[[CMDB_class_name]] attribute[[CMDB_attribute_name]] to_DB_class
[com.mercury.topaz.fcmbd.adapters.dbAdapter.dal.
transform.impl.GenericEnumTransformer(generic-enum-transformer-
example.xml)]
from_DB_class
[com.mercury.topaz.fcmbd.adapters.dbAdapter.dal.transform.impl.
GenericEnumTransformer(generic-enum-transformer-example.xml)]
```

entity. The entity name as it appears in the `orm.xml` file.

attribute. The attribute name as it appears in the `orm.xml` file.

to_DB_class. The full, qualified name of a class that implements the interface **com.mercury.topaz.fcmbd.adapters.dbAdapter.dal.transform.FcmbdDalTransformerToExternalDB**. The elements in the parenthesis are given to this class constructor. Use this converter to transform RTSM values to database values, for example, to append the suffix of **.com** to each node name.

from_DB_class. The full, qualified name of a class that implements the **com.mercury.topaz.fcmbd.adapters.dbAdapter.dal.transform.FcmbdDalTransformerFromExternalDB** interface. The elements in the parenthesis are given to this class constructor. Use this converter to transform database values to RTSM values, for example, to append the suffix of **.com** to each node name.

For details, see ["Out-of-the-Box Converters"](#) on page 158.

The discriminator.properties File

This file maps each supported CI type (that is also used as a discriminator value in orm.xml) to a comma-separated list of possible corresponding values of the discriminator column, or a condition to match possible values of the discriminator column.

If a condition is used, use the syntax: `like (condition)`, where `condition` is a string that can contain the following wildcards:

- `%` (percent sign) - allows you to match any string of any length (including a zero length string)
- `_` (underscore) - allows you to match a single character

For example, `like (%unix%)` will match `unix`, `linux`, `unix-aix`, and so on. Like conditions may only be applied to string columns.

You can also have a single discriminator value mapped to any value that does not belong to another discriminator by stating `'all-other'`.

If the adapter you are creating uses discriminator capabilities, you must define all the discriminator values in the **discriminator.properties** file.

Example of Discriminator Mapping:

For example, the adapter supports the CI types `node`, `nt`, and `unix`, and the database contains a single table named `t_nodes` that contains a column called **type**. If the type is 10001, the row represents a node; if the type is 10004, it represents a unix machine, and so on. The **discriminator.properties** file might look like this:

```
node=10001, 10005
nt=10002,10003
unix=2%
mainframe=all-other
```

The **orm.xml** file includes the following code:

```
<entity class="generic_db_adapter.node" >
  <table name="t_nodes" />
  ...
  <inheritance strategy="SINGLE_TABLE" />
  <discriminator-value>node</discriminator-value>
  <discriminator-column name="type" />
  ...
</entity>
<entity class="generic_db_adapter.nt" name="nt">
  <discriminator-value>nt</discriminator-value>
  <attributes>
</entity>
<entity class="generic_db_adapter.unix" name="unix">
  <discriminator-value>unix</discriminator-value>
  <attributes>
</entity>
```

The `discriminator_column` attribute is then calculated as follows:

- If **type** contains 10002 or 10003 for a certain entry, the entry is mapped to the **nt** CIT.
- If **type** contains 10001 or 10005 for a certain entry, the entry is mapped to the **node** CIT.
- If **type** starts with 2 for a certain entry, the entry is mapped to the **unix** CIT.
- Any other value in the **type** column is mapped to the **mainframe** CIT.

Note: The **node** CIT is also the parent of **nt** and **unix**.

The replication_config.txt File

This file contains a comma-separated list of CI and relationship types whose property conditions are supported by the replication plugin. For details, see "Plugins" on page 162.

The fixed_values.txt File

This file enables you to configure fixed values for specific attributes of certain CITs. In this way, each of these attributes can be assigned a fixed value that is not stored in the database.

The file should contain zero or more entries of the following format:

```
entity[<entityName>] attribute[<attributeName>] value[<value>]
```

For example:

```
entity[ip_address] attribute[ip_domain] value[DefaultDomain]
```

The file also supports a list of constants. To define a constants list, use the following syntax:

```
entity[<entityName>] attribute[<attributeName>] value[{<Val1>, <Val2>,  
<Val3>, ... }]
```

The persistence.xml File

This file is used to override the default Hibernate settings and to add support for database types that are not out of the box (OOB database types are Oracle Server, Microsoft SQL Server, and MySQL).

If you need to support a new database type, make sure that you supply a connection pool provider (the default is c3p0) and a JDBC driver for your database (put the *.jar files in the adapter folder).

To see all available Hibernate values that can be changed, check the **org.hibernate.cfg.Environment** class (for details, refer to <http://www.hibernate.org>).

Example of the persistence.xml File:

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi=  
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=  
"http://java.sun.com/xml/ns/persistence  
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"  
version="1.0">  
  <!-- Don't change this value -->  
  <persistence-unit name="GenericDBAdapter">  
    <properties>
```

```
        <!-- Don't change this value -->
        <property name="hibernate.archive.autodetection"
value="class,
        hbm" />
        <!--The driver class name"/-->
        <property name="hibernate.connection.driver_class"
value="com.mercury.
        jdbc.MercOracleDriver" />
        <!--The connection url"/-->
        <property name="hibernate.connection.url"
value="jdbc:mercury:oracle:
        //artist:1521;sid=cmdb2" />
        <!--DB login credentials"/-->
        <property name="hibernate.connection.username"
value="CMDB" />
        <property name="hibernate.connection.password"
value="CMDB" />
        <!--connection pool properties"/-->
        <property name="hibernate.c3p0.min_size" value="5" />
        <property name="hibernate.c3p0.max_size" value="20" />
        <property name="hibernate.c3p0.timeout" value="300" />
        <property name="hibernate.c3p0.max_statements" value="50"
/>
        <property name="hibernate.c3p0.idle_test_period"
value="3000" />
        <!--The dialect to use-->
        <property name="hibernate.dialect"
value="org.hibernate.dialect.
        OracleDialect" />
    </properties>
</persistence-unit>
</persistence>
```

Out-of-the-Box Converters

You can use the following converters (transformers) to convert federated queries and replication jobs to and from database data.

This section includes the following topics:

- ["Out-of-the-Box Converters" above](#)
- ["The SuffixTransformer Converter" on page 160](#)
- ["The PrefixTransformer Converter" on page 161](#)
- ["The BytesToStringTransformer Converter" on page 161](#)
- ["The StringDelimitedListTransformer Converter" on page 161](#)

The enum-transformer Converter

This converter uses an XML file that is given as an input parameter.

The XML file maps between hard-coded RTSM values and database values (enums). If one of the values does not exist, you can choose to return the same value, return null, or throw an exception.

The transformer performs a comparison between two strings using a case sensitive, or a case insensitive method. The default behavior is case sensitive. To define it as case insensitive use: `case-sensitive="false"` in the `enum-transformer` element.

Use one XML mapping file for each entity attribute.

Note: This converter can be used for both the `to_DB_class` and `from_DB_class` fields in the `transformations.txt` file.

Input File XSD:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">

  <xs:element name="enum-transformer">

    <xs:complexType>

      <xs:sequence>

        <xs:element ref="value" minOccurs="0"
maxOccurs="unbounded"/>

      </xs:sequence>

      <xs:attribute name="db-type" use="required">

        <xs:simpleType>

          <xs:restriction base="xs:string">

            <xs:enumeration value="integer"/>

            <xs:enumeration value="long"/>

            <xs:enumeration value="float"/>

          </xs:restriction>

        </xs:simpleType>

      </xs:attribute>

    </xs:complexType>

  </xs:element>

</xs:schema>
```

```
        <xs:enumeration value="double"/>
        <xs:enumeration value="boolean"/>
        <xs:enumeration value="string"/>
        <xs:enumeration value="date"/>
        <xs:enumeration value="xml"/>
        <xs:enumeration value="bytes"/>
    </xs:restriction>
</xs:simpleType>
</xs:attribute>
<xs:attribute name="cmdb-type" use="required">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:enumeration value="integer"/>
            <xs:enumeration value="long"/>
            <xs:enumeration value="float"/>
            <xs:enumeration value="double"/>
            <xs:enumeration value="boolean"/>
            <xs:enumeration value="string"/>
            <xs:enumeration value="date"/>
            <xs:enumeration value="xml"/>
            <xs:enumeration value="bytes"/>
        </xs:restriction>
    </xs:simpleType>
</xs:attribute>
<xs:attribute name="non-existing-value-action"
use="required">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:enumeration value="return-null"/>
            <xs:enumeration value="return-original"/>
            <xs:enumeration value="throw-exception"/>
        </xs:restriction>
    </xs:simpleType>
```

```
</xs:attribute>

<xs:attribute name="case-sensitive" use="optional">

  <xs:simpleType>

    <xs:restriction base="xs:boolean">

      </xs:restriction>

    </xs:simpleType>

  </xs:attribute>

</xs:complexType>

</xs:element>

<xs:element name="value">

  <xs:complexType>

    <xs:attribute name="cmdb-value" type="xs:string"
use="required"/>

    <xs:attribute name="external-db-value" type="xs:string"
use="required"/>

    <xs:attribute name="is-cmdb-value-null" type="xs:boolean"
use="optional"/>

    <xs:attribute name="is-db-value-null" type="xs:boolean"
use="optional"/>

  </xs:complexType>

</xs:element>

</xs:schema>
```

Example of Converting 'sys' Value to 'System' Value:

In this example, `sys` value in the RTSM is transformed into `System` value in the federated database, and `System` value in the federated database is transformed into `sys` value in the RTSM.

If the value does not exist in the XML file (for example, the string `demo`), the converter returns the same input value it receives.

```
<enum-transformer CMDB-type="string" DB-type="string" non-existing-
value-action="return-original"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=" ../META-CONF/generic-enum-
transformer.xsd">
  <value CMDB-value="sys" external-DB-value="System" />
</enum-transformer>
```

The SuffixTransformer Converter

This converter is used to add or remove suffixes from the RTSM or federated database source value.

There are two implementations:

- **com.mercury.topaz.fcldb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdbAddSuffixTransformer.** Adds the suffix (given as input) when converting from federated database value to RTSM value and removes the suffix when converting from RTSM value to federated database value.
- **com.mercury.topaz.fcldb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdbRemoveSuffixTransformer.** Removes the suffix (given as input) when converting from federated database value to RTSM value and adds the suffix when converting from RTSM value to federated database value.

The PrefixTransformer Converter

This converter is used to add or remove a prefix from the RTSM or federated database value.

There are two implementations:

- **com.mercury.topaz.fcldb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdbAddPrefixTransformer.** Adds the prefix (given as input) when converting from federated database value to RTSM value and removes the prefix when converting from RTSM value to federated database value.
- **com.mercury.topaz.fcldb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdbRemovePrefixTransformer.** Removes the prefix (given as input) when converting from federated database value to RTSM value and adds the prefix when converting from RTSM value to federated database value.

The BytesToStringTransformer Converter

This converter is used to convert byte arrays in the RTSM to their string representation in the federated database source.

The converter is:

com.mercury.topaz.fcldb.adapters.dbAdapter.dal.transform.impl.CmdbToAdapterBytesToStringTransformer.

The StringDelimitedListTransformer Converter

This converter is used to transform a single string list to an integer/string list in the RTSM.

The converter is: **com.mercury.topaz.fcldb.adapters.dbAdapter.dal.transform.impl.StringDelimitedListTransformer.**

Plugins

The generic database adapter supports the following plugins:

- An optional plugin for full topology synchronization.
- An optional plug-in for synchronizing changes in topology. If no plug-in for synchronizing changes is implemented, it is possible to perform a differential synchronization, but that synchronization will actually be a full one.
- An optional plugin for synchronizing layout.
- An optional plugin to retrieve supported queries for synchronization. If this plugin is not defined, all TQL names are returned.
- An internal, optional plugin to change the TQL definition and TQL result.
- An internal, optional plugin to change a layout request and CIs result.
- An internal, optional plugin to change a layout request and relationships result.

For details about implementing and deploying plugins, see ["Implement a Plugin" on page 123](#).

Configuration Examples

This section gives examples of configurations.

This section includes the following topics:

- ["Use Case" below](#)
- ["Single Node Reconciliation" below](#)
- ["Two Node Reconciliation" on page 166](#)
- ["Using a Primary Key that Contains More Than One Column" on page 168](#)
- ["Using Transformations" on page 170](#)

Use Case

Use case. A TQL query is:

node > (composition) > card

where:

- **node** is the RTSM entity
- **card** is the federated database source entity
- **composition** is the relationship between them

The example is run against the **ED** database. **ED nodes** are stored in the **Device** table and **card** is stored in the **hwCards** table. In the following examples, **card** is always mapped in the same manner.

Single Node Reconciliation

In this example the reconciliation is run against the **name** property.

Simplified Definition

The reconciliation is done by **node** and it is emphasized by the special tag **CMDB-class**.

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=" ../META-
CONF/simplifiedConfiguration.xsd">
  <CMDB-class CMDB-class-name="node" default-table-name="Device">
    <primary-key column-name="Device_ID" />
    <reconciliation-by-single-node>
      <or>
        <attribute CMDB-attribute-name="name" column-
name="Device_Name" />
      </or>
    </reconciliation-by-single-node>
  </CMDB-class>
```

```
<class CMDB-class-name="card" default-table-name="hwCards"
connected-CMDB-class-name="node" link-class-name="composition">
  <foreign-primary-key column-name="Device_ID" CMDB-class-
primary-key-column="Device_ID
  <primary-key column-name="hwCards_Seq" />
  <attribute CMDB-attribute-name="card_class" column-
name="hwCardClass" />
  <attribute CMDB-attribute-name="card_vendor" column-
name="hwCardVendor" />
  <attribute CMDB-attribute-name="card_name" column-
name="hwCardName" />
</class>
</generic-DB-adapter-config>
```

Advanced Definition

The orm.xml File

Pay attention to the addition of the relationship mapping. For details, see the definition section in ["The orm.xml File" on page 143](#).

Example of the orm.xml File:

```
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"
xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/
persistence/orm http://java.sun.com/xml/ns/persistence/orm_1_0.xsd"
version="1.0">
  <description>Generic DB adapter orm</description>
  <package>generic_db_adapter</package>
  <entity class="generic_db_adapter.node" >
    <table name="Device"/>
    <attributes>
      <id name="id1">
        <column name="Device_ID" insertable="false"
          updatable="false"/>
        <generated-value strategy="TABLE"/>
      </id>
      <basic name="name">
        <column name="Device_Name"/>
      </basic>
    </attributes>
  </entity>
  <entity class="generic_db_adapter.card" >
    <table name="hwCards"/>
    <attributes>
      <id name="id1">
        <column name="hwCards_Seq" insertable="false"
          updatable="false"/>
      </id>
    </attributes>
  </entity>
</entity-mappings>
```

```

        <generated-value strategy="TABLE"/>
    </id>
    <basic name="card_class">
        <column name="hwCardClass" insertable="false"
            updatable="false"/>
    </basic>
    <basic name="card_vendor">
        <column name="hwCardVendor" insertable="false"
            updatable="false"/>
    </basic>
    <basic name="card_name">
        <column name="hwCardName" insertable="false"
            updatable="false"/>
    </basic>
</attributes>
</entity>
<entity class="generic_db_adapter.node_composition_card" >
    <table name="hwCards"/>
    <attributes>
        <id name="id1">
            <column name="hwCards_Seq" insertable="false"
                updatable="false"/>
            <generated-value strategy="TABLE"/>
        </id>
        <many-to-one name="end1" target-entity="node">
            <join-column name="Device_ID" insertable="false"
                updatable="false"/>
        </many-to-one>
        <one-to-one name="end2" target-entity="card">
            <join-column name="hwCards_Seq"
                referenced-column-name="hwCards_Seq" insertable=
                "false" updatable="false"/>
        </one-to-one>
    </attributes>
</entity>
</entity-mappings>

```

The reconciliation_types.txt File

For details, see ["The reconciliation_types.txt file" on page 152.](#)

node

The reconciliation_rules.txt File

For details, see ["The reconciliation_rules.txt File \(for backwards compatibility\)" on page 152.](#)

multinode[node] expression[node.name]

The transformation.txt File

This file remains empty as no values need to be converted in this example.

Two Node Reconciliation

In this example, reconciliation is calculated according to the `name` property of `node` and of `ip_address` with different variations.

The reconciliation TQL query is **node > (containment) > ip_address**.

Simplified Definition

The reconciliation is by name of node OR of ip_address:

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=" ../META-
CONF/simplifiedConfiguration.xsd">
    <CMDB-class CMDB-class-name="node" default-table-name="Device">
        <primary-key column-name="Device_ID" />
        <reconciliation-by-two-nodes connected-node-CMDB-class-
name="ip_address" CMDB-link-type="containment">
            <or>
                <attribute CMDB-attribute-name="name" column-
name="Device_Name" />
                <connected-node-attribute CMDB-attribute-name="name"
column-name="Device_PREFERREDIPAddress" />
            </or>
        </reconciliation-by-two-nodes>
    </CMDB-class>
    <class CMDB-class-name="card" default-table-name="hwCards"
connected-CMDB-class-name="node" link-class-name="containment">
        <foreign-primary-key column-name="Device_ID" CMDB-class-
primary-key-column="Device_ID" />
        <primary-key column-name="hwCards_Seq" />
        <attribute CMDB-attribute-name="card_class" column-
name="hwCardClass" />
        <attribute CMDB-attribute-name="card_vendor" column-
name="hwCardVendor" />
        <attribute CMDB-attribute-name="card_name" column-
name="hwCardName" />
    </class>
</generic-DB-adapter-config>
```

The reconciliation is name of node AND of ip_address:

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=" ../META-
CONF/simplifiedConfiguration.xsd">
    <CMDB-class CMDB-class-name="node" default-table-name="Device">
        <primary-key column-name="Device_ID" />
        <reconciliation-by-two-nodes connected-node-CMDB-class-
```

```

name="ip_address" CMDB-link-type="containment">
    <and>
        <attribute CMDB-attribute-name="name" column-
name="Device_Name" />
        <connected-node-attribute CMDB-attribute-name="name"
column-name="Device_PREFERREDIPAddress" />
    </and>
</reconciliation-by-two-nodes>
</CMDB-class>
<class CMDB-class-name="card" default-table-name="hwCards"
connected-CMDB-class-name="node" link-class-name="containment">
    <foreign-primary-key column-name="Device_ID" CMDB-class-
primary-key-column="Device_ID" />
    <primary-key column-name="hwCards_Seq" />
    <attribute CMDB-attribute-name="card_class" column-
name="hwCardClass" />
    <attribute CMDB-attribute-name="card_vendor" column-
name="hwCardVendor" />
    <attribute CMDB-attribute-name="card_name" column-
name="hwCardName" />
</class>
</generic-DB-adapter-config>

```

The reconciliation is by name of ip_address:

```

<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../../META-
CONF/simplifiedConfiguration.xsd">
    <CMDB-class CMDB-class-name="node" default-table-name="Device">
        <primary-key column-name="Device_ID" />
        <reconciliation-by-two-nodes connected-node-CMDB-class-
name="ip_address" CMDB-link-type="containment">
            <or>
                <connected-node-attribute CMDB-attribute-name="name"
column-name="Device_PREFERREDIPAddress" />
            </or>
        </reconciliation-by-two-nodes>
    </CMDB-class>
    <class CMDB-class-name="card" default-table-name="hwCards"
connected-CMDB-class-name="node" link-class-name="containment">
        <foreign-primary-key column-name="Device_ID" CMDB-class-
primary-key-column="Device_ID" />
        <primary-key column-name="hwCards_Seq" />
        <attribute CMDB-attribute-name="card_class" column-
name="hwCardClass" />
        <attribute CMDB-attribute-name="card_vendor" column-
name="hwCardVendor" />
        <attribute CMDB-attribute-name="card_name" column-
name="hwCardName" />
    </class>
</generic-DB-adapter-config>

```

```
</class>  
</generic-DB-adapter-config>
```

Advanced Definition

The orm.xml File

Since the reconciliation expression is not defined in this file, the same version should be used for any reconciliation expression.

The reconciliation_types.txt File

For details, see ["The reconciliation_types.txt file" on page 152.](#)

```
node
```

The reconciliation_rules.txt File

For details, see ["The reconciliation_rules.txt File \(for backwards compatibility\)" on page 152.](#)

```
multinode[node] expression[ip_address.name OR node.name] end1_type  
[node] end2_type[ip_address] link_type[containment]  
  
multinode[node] expression[ip_address.name AND node.name] end1_type  
[node] end2_type[ip_address] link_type[containment]  
  
multinode[node] expression[ip_address.name] end1_type[node] end2_type  
[ip_address] link_type[containment]
```

The transformation.txt File

This file remains empty as no values need to be converted in this example.

Using a Primary Key that Contains More Than One Column

If the primary key is composed of more than one column, the following code is added to the XML definitions:

Simplified Definition

There is more than one primary key tag and for each column there is a tag.

```
<class CMDB-class-name="card" default-table-name="hwCards"  
connected-CMDB-class-name="node" link-class-name="containment">  
  <foreign-primary-key column-name="Device_ID" CMDB-class-  
primary-key-column="Device_ID" />  
  <primary-key column-name="Device_ID" />  
  <primary-key column-name="hwBusesSupported_Seq" />  
  <primary-key column-name="hwCards_Seq" />  
  <attribute CMDB-attribute-name="card_class" column-  
name="hwCardClass" />  
  <attribute CMDB-attribute-name="card_vendor" column-  
name="hwCardVendor" />  
  <attribute CMDB-attribute-name="card_name" column-
```



```
name="hwCardName" />
</class>
```

Advanced Definition

The orm.xml File

A new `id` entity is added that maps to the primary key columns. Entities that use this `id` entity must add a special tag.

If you use a foreign key (join-column tag) for such a primary key, you must map between each column in the foreign key to a column in the primary key.

For details, see ["The orm.xml File" on page 143](#).

Example of the orm.xml File:

```
<entity class="generic_db_adapter.card" >
  <table name="hwCards" />
  <attributes>
    <id name="id1">
      <column name="Device_ID" insertable="false"
updateable="false" />
      <generated-value strategy="TABLE" />
    </id>
    <id name="id2">
      <column name="hwBusesSupported_Seq" insertable="false"
updateable="false" />
      <generated-value strategy="TABLE" />
    </id>
    <id name="id3">
      <column name="hwCards_Seq" insertable="false"
updateable="false" />
      <generated-value strategy="TABLE" />
    </id>
  </attributes>
</entity>

<entity class="generic_db_adapter.node_containment_card" >
  <table name="hwCards" />
  <attributes>
    <id name="id1">
      <column name="Device_ID" insertable="false"
updateable="false" />
      <generated-value strategy="TABLE" />
    </id>
    <id name="id2">
      <column name="hwBusesSupported_Seq" insertable="false"
updateable="false" />
      <generated-value strategy="TABLE" />
    </id>
    <id name="id3">
      <column name="hwCards_Seq" insertable="false"
updateable="false" />
      <generated-value strategy="TABLE" />
    </id>
  </attributes>
</entity>
```

```
</id>
<many-to-one name="end1" target-entity="node">
  <join-column name="Device_ID" insertable="false"
updatable="false" />
</many-to-one>
<one-to-one name="end2" target-entity="card">
  <join-column name="Device_ID" referenced-column-
name="Device_ID" insertable="false" updatable="false" />
  <join-column name="hwBusesSupported_Seq" referenced-
column-name="hwBusesSupported_Seq" insertable="false"
updatable="false" />
  <join-column name="hwCards_Seq" referenced-column-
name="hwCards_Seq" insertable="false" updatable="false" />
</one-to-one>
</attributes>
</entity>
</entity-mappings>
```

Using Transformations

In the following example, the generic **enum** transformer is converted from values 1, 2, 3 to values a, b, c respectively in the **name** column.

The mapping file is `generic-enum-transformer-example.xml`.

```
<enum-transformer CMDB-type="string" DB-type="string" non-existing-
value-action="return-original"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../META-CONF/generic-enum-
transformer.xsd">
  <value CMDB-value="1" external-DB-value="a" />
  <value CMDB-value="2" external-DB-value="b" />
  <value CMDB-value="3" external-DB-value="c" />
</enum-transformer>
```

Simplified Definition

```
<CMDB-class CMDB-class-name="node" default-table-name="Device">
  <primary-key column-name="Device_ID" />
  <reconciliation-by-two-nodes connected-node-CMDB-class-
name="ip_address"
  CMDB-link-type="containment">
    <or>
      <attribute CMDB-attribute-name="name" column-
name="Device_Name"
        from-CMDB-
converter="com.mercury.topaz.fcmbd.adapters.dbAdapter.dal.
transform.impl.GenericEnumTransformer(generic-enum-
transformer-example.
xml)" to-CMDB-
converter="com.mercury.topaz.fcmbd.adapters.dbAdapter.dal.
```

```
        transform.impl.GenericEnumTransformer(generic-enum-  
transformer-example.  
        xml)" />  
        <connected-node-attribute CMDB-attribute-name="name"  
        column-name="Device_PREFERREDIPAddress" />  
    </or>  
</reconciliation-by-two-nodes>  
</CMDB-class>
```

Advanced Definition

There is a change only to the **transformation.txt** file.

The transformation.txt File

Make sure that the attribute names and entity names are the same as in the `orm.xml` file.

```
entity[node] attribute[name]  
to_DB_class  
[com.mercury.topaz.fcmbd.adapters.dbAdapter.dal.transform.impl.  
GenericEnumTransformer(generic-enum-transformer-example.xml)] from_DB_  
class  
[com.mercury.topaz.fcmbd.adapters.dbAdapter.dal.transform.impl.  
GenericEnumTransformer(generic-enum-transformer-example.xml)]
```

Adapter Log Files

To understand the calculation flows and adapter lifecycle, and to view debug information, you can consult the following log files.

This section includes the following topics:

- "Log Levels" below
- "Log Locations" below

Log Levels

You can configure the log level for each of the logs.

In a text editor, open the
<HP BSM root directory>\odb\conf\log\fcmdb.gdba.properties file.

The default log level is **ERROR**:

```
#loglevel can be any of DEBUG INFO WARN ERROR FATAL  
loglevel=ERROR
```

- To increase the log level for all log files, change **loglevel=ERROR** to **loglevel=DEBUG** or **loglevel=INFO**.
- To change the log level for a specific file, change the specific **log4j** category line accordingly. For example, to change the log level of **fcmdb.gdba.dal.sql.log** to **INFO**, change:

```
log4j.category.fcmdb.gdba.dal.SQL=${loglevel},  
fcmdb.gdba.dal.SQL.appender
```

to:

```
log4j.category.fcmdb.gdba.dal.SQL=INFO, fcmdb.gdba.dal.SQL.appender
```

Log Locations

The log files are located in the **<HP BSM root directory>\odb\runtime\log** directory.

- **Fcmdb.gdba.log**

The adapter lifecycle log. Gives details about when the adapter started or stopped, and which CITs are supported by this adapter.

Consult for initiation errors (adapter load/unload).

- **fcmdb.log**

Consult for exceptions.

- **cmdb.log**

Consult for exceptions.

- **Fcmdb.gdba.mapping.engine.log**

The mapping engine log. Gives details about the reconciliation TQL query that the mapping engine uses, and the reconciliation topologies that are compared during the connect phase.

Consult this log when a TQL query gives no results even though you know there are relevant CIs in the database, or the results are unexpected (check the reconciliation).

- **Fcmdb.gdba.TQL.log**

The TQL log. Gives details about the TQL queries and their results.

Consult this log when a TQL query does not return results and the mapping engine log shows that there are no results in the federated data source.

- **Fcmdb.gdba.dal.log**

The DAL lifecycle log. Gives details about CIT generation and database connection details.

Consult this log when you cannot connect to the database or when there are CITs or attributes that are not supported by the query.

- **Fcmdb.gdba.dal.command.log**

The DAL operations log. Gives details about internal DAL operations that are called. (This log is similar to `cmdb.dal.command.log`).

- **Fcmdb.gdba.dal.SQL.log**

The DAL SQL queries log. Gives details about called JPAQLs (object oriented SQL queries) and their results.

Consult this log when you cannot connect to the database or when there are CITs or attributes that are not supported by the query.

- **Fcmdb.gdba.hibernate.log**

The Hibernate log. Gives details about the SQL queries that are run, the parsing of each JPAQL to SQL, the results of the queries, data regarding Hibernate caching, and so on. For details on Hibernate, see ["Hibernate as JPA Provider" on page 103](#).

External References

For details on the JavaBeans 3.0 specification, see

<http://jcp.org/aboutJava/communityprocess/final/jsr220/index.html>.

Troubleshooting and Limitations

This section describes troubleshooting and limitations for the generic database adapter.

General Limitations

- SQL Server NTLM authentication is not supported.
- When you update an adapter package, use Notepad++, UltraEdit, or some other third-party text editor rather than Notepad (any version) from Microsoft Corporation to edit the template files. This prevents the use of special symbols, which cause the deployment of the prepared package to fail.

JPA Limitations

- All tables must have a primary key column.
- RTSM class attribute names must follow the JavaBeans naming convention (for example, names must start with lower case letters).
- Two CIs that are connected with one relationship in the class model must have direct association in the database (for example, if `node` is connected to `ticket` there must be a foreign key or linkage table that connects them).
- Several tables that are mapped to the same CIT must share the same primary key table.

Functional Limitations

- You cannot create a manual relationship between the RTSM and federated CITs. To be able to define virtual relationships, a special relationship logic must be defined (it can be based on properties of the federated class).
- Federated CITs cannot be trigger CITs in an impact rule, but they can be included in an impact analysis TQL query.
- A federated CIT can be part of an enrichment TQL, but cannot be used as the node on which enrichment is performed (you cannot add, update, or delete the federated CIT).
- Using a class qualifier in a condition is not supported.
- Subgraphs are not supported.
- Compound relationships are not supported.
- The external CI `RTSMid` is composed from its primary key and not its key attributes.
- A column of type `bytes` cannot be used as a primary key column in Microsoft SQL Server.
- TQL query calculation fails if attribute conditions that are defined on a federated node have not had their names mapped in the `orm.xml` file.
- The Generic DB Adapter does not support Windows Authentication for SQL Server.

Chapter 6

Developing Java Adapters

This chapter includes:

Federation Framework Overview	177
Adapter and Mapping Interaction with the Federation Framework	182
Federation Framework for Federated TQL Queries	183
Interactions between the Federation Framework, Server, Adapter, and Mapping Engine	185
Federation Framework Flow for Population	194
Adapter Interfaces	196
Debug Adapter Resources	198
Add an Adapter for a New External Data Source	199
Implement the Mapping Engine	206
Create a Sample Adapter	208
XML Configuration Tags and Properties	208

Federation Framework Overview

Note:

- The term **relationship** is equivalent to the term **link**.
- The term **CI** is equivalent to the term **object**.
- A graph is a collection of nodes and links.
- For a glossary of definitions and terms, see Glossary in the *RTSM Administration Guide*.

The Federation Framework functionality uses an API to retrieve information from federated sources. The Federation Framework provides three main capabilities:

- **Federation** on the fly. All queries are run over original data repositories and results are built on the fly in the RTSM.
- **Population**. Populates data (topological data and CI properties) to the RTSM from an external data source.
- **Data Push**. Pushes data (topological data and CI properties) from the local RTSM to a remote data source.

All action types require an adapter for each data repository, which can provide the specific capabilities of the data repository and retrieve and/or update the required data. Every request to the data repository is made through its adapter.

This section also includes the following topics:

- ["Federation on the Fly" below](#)
- ["Data Push" on page 179](#)
- ["Population" on page 179](#)

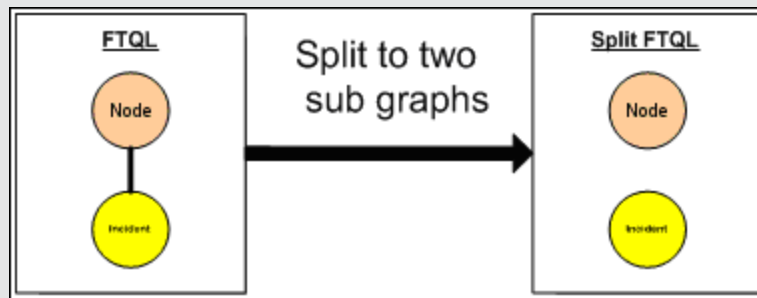
Federation on the Fly

Federated TQL queries enables data retrieval from any external data repository without replicating its data.

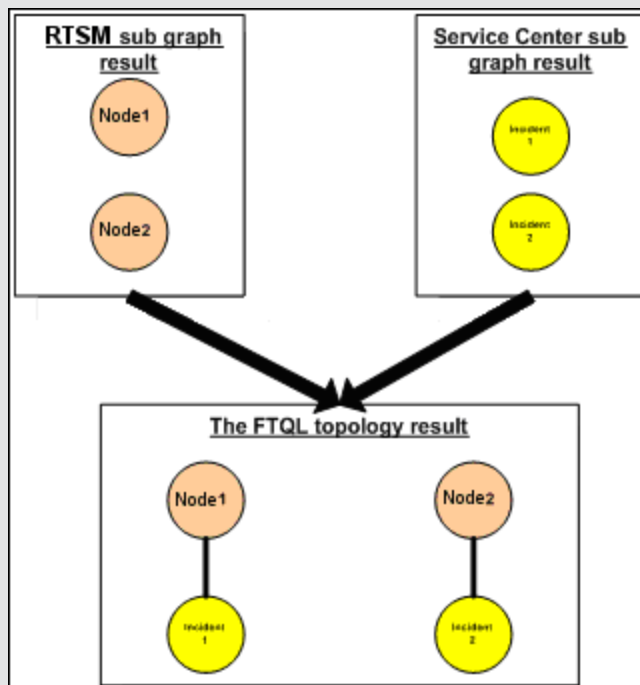
A federated TQL query uses adapters that represent external data repositories, to create appropriate external relationships between CIs from different external data repositories and the BSM CIs.

Example of Federation-on-the-Fly Flow:

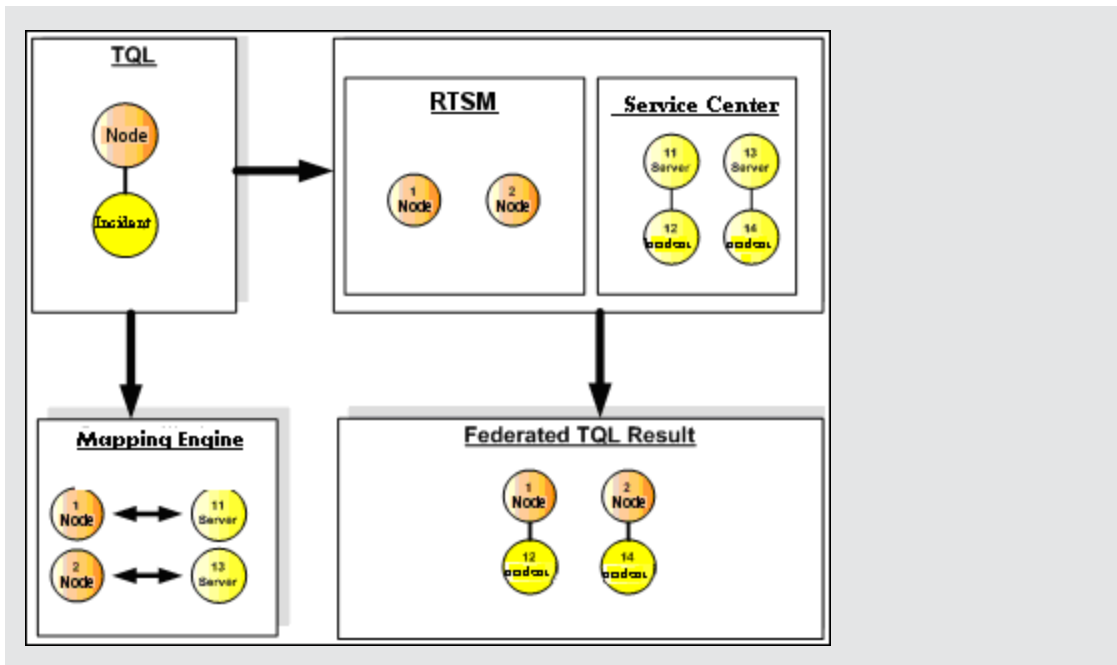
1. The Federation Framework splits a federated TQL query into several subgraphs, where all nodes in a subgraph refer to the same data repository. Each subgraph is connected to the other subgraphs by a virtual relationship (but itself contains no virtual relationships).



2. After the federated TQL query is split into subgraphs, the Federation Framework calculates each subgraph's topology and connects two appropriate subgraphs by creating virtual relationships between the appropriate nodes.



3. After the federated TQL topology is calculated, the Federation Framework retrieves a layout for the topology result.



Data Push

You use the data push flow to synchronize data from your current local RTSM to a remote service or target data repository.

In data push, data repositories are divided into two categories: source (local RTSM) and target. Data is retrieved from the source data repository and updated to the target data repository. The data push process is based on query names, meaning that data is synchronized between the source (local RTSM) and target data repositories, and is retrieved by a TQL query name from the local RTSM.

The data push process flow includes the following steps:

1. Retrieving the topology result with signatures from the source data repository.
2. Comparing the new results with the previous results.
3. Retrieving a full layout (that is, all CI properties) of CIs and relationships, for changed results only.
4. Updating the target data repository with the received full layout of CIs and relationships. If any CIs or relationships are deleted in the source data repository and the query is exclusive, the replication process removes the CIs or relationships in the target data repository as well.

The RTSM has 2 hidden data sources (**hiddenRMIDataSource** and **hiddenChangesDataSource**), which are always the 'source' data source in data push flows. To implement a new adapter for data push flows, you only have to implement the 'target' adapter.

Population

You use the population flow to populate the RTSM with data from external sources.

The flow always uses one 'source' data source to retrieve the data, and pushes the retrieved data to the Probe in a similar process to the flow of a discovery job.

To implement a new adapter for population flows, you only have to implement the source adapter, since the Data Flow Probe acts as the target.

The adapter in the population flow is executed on the Probe. Debugging and logging should be done on the Probe and not on the RTSM.

The population flow is based on query names, that is, data is synchronized between the source data repository and the Data Flow Probe, and is retrieved by a query name in the source data repository. For example, in BSM, the query name is the name of the TQL query. However, in another data repository the query name can be a code name that returns data. The adapter is designed to correctly handle the query name.

Each job can be defined as an exclusive job. This means that the CIs and relationships in the job results are unique in the local RTSM, and no other query can bring them to the target. The adapter of the source data repository supports specific queries, and can retrieve the data from this data repository. The adapter of the target data repository enables the update of retrieved data on this data repository.

SourceDataAdapter Flow

- Retrieves the topology result with signatures from the source data repository.
- Compares the new results with the previous results.
- Retrieves a full layout (that is, all CI properties) of CIs and relationships, for changed results only.
- Updates the target data repository with the received full layout of CIs and relationships. If any CIs or relationships are deleted in the source data repository and the query is exclusive, the replication process removes the CIs or relationships in the target data repository as well.

SourceChangesDataAdapter Flow

- Retrieves the topology result that occurred since the last date given.
- Retrieves a full layout (that is, all CI properties) of CIs and relationships, for changed results only.
- Updates the target data repository with the received full layout of CIs and relationships. If any CIs or relationships are deleted in the source data repository and the query is exclusive, the replication process removes the CIs or relationships in the target data repository as well.

PopulateDataAdapter Flow

- Retrieves the full topology with requested layout result.
- Uses the topology chunk mechanism to retrieve the data in chunks.
- The probe filters out any data that was already brought in earlier runs
- Updates the target data repository with the received layout of CIs and relationships. If any CIs or relationships are deleted in the source data repository and the query is exclusive, the replication process removes the CIs or relationships in the target data repository as well.

PopulateChangesDataAdapter Flow

- Retrieves the topology with requested layout result that has changes since the last run.
- Uses the topology chunk mechanism to retrieve the data in chunks.
- The probe filters out any data that was already brought in earlier runs (including this flow).
- Updates the target data repository with the received layout of CIs and relationships. If any CIs or relationships are deleted in the source data repository and the query is exclusive, the replication process removes the CIs or relationships in the target data repository as well.

Adapter and Mapping Interaction with the Federation Framework

An adapter is an entity in BSM that represents external data (data that is not saved in BSM). In federated flows, all interactions with external data sources are performed through adapters. The Federation Framework interaction flow and adapter interfaces are different for replication and for federated TQL queries.

This section also includes the following topics:

- ["Adapter Lifecycle" below](#)
- ["Adapter assist Methods" below](#)

Adapter Lifecycle

An adapter instance is created for each external data repository. The adapter begins its lifecycle with the first action applied to it (such as, `calculate TQL` or `retrieve/update data`). When the **start** method is called, the adapter receives environmental information, such as the data repository configuration, logger, and so on. The adapter lifecycle ends when the data repository is removed from the configuration, and the **shutdown** method is called. This means that the adapter is stateful and can contain the connection to the external data repository if it is required.

Adapter assist Methods

The adapter has several `assist` methods that can add external data repository configurations. These methods are not part of the adapter lifecycle and create a new adapter each time they are called.

- The first method tests the connection to the external data repository for a given configuration. `testConnection` can be executed either on the BSM server or the Data Flow Probe, depending on the type of adapter.
- The second method is relevant only for the source adapter and returns the supported queries for replication. (This method is executed on the Probe only.)
- The third method is relevant only for federation and population flows, and returns supported external classes by the external data repository. (This method is executed on the BSM server.)

All these methods are used when you create or view integration configurations.

Federation Framework for Federated TQL Queries

This section includes the following topics:

- ["Definitions and Terms" below](#)
- ["Mapping Engine" below](#)
- ["Federated Adapter" below](#)

See ["Interactions between the Federation Framework, Server, Adapter, and Mapping Engine"](#) on [page 185](#) for diagrams illustrating the interactions between the Federation Framework, BSM, adapter, and Mapping Engine.

Definitions and Terms

Reconciliation data. The rule for matching CIs of the specified type that are received from the RTSM and the external data repository. The reconciliation rule can be of three types:

- **ID reconciliation.** This can be used only if the external data repository contains the RTSM ID of reconciliation objects.
- **Property reconciliation.** This is used when the matching can be done by properties of the reconciliation CI type only.
- **Topology reconciliation.** This is used when you need the properties of additional CITs (not only of the reconciliation CIT) to perform a match on reconciliation CIs. For example, you can perform reconciliation of the node type by the `name` property that belongs to the `ip_address` CIT.

Reconciliation object. The object is created by the adapter according to received reconciliation data. This object should refer to an external CI and is used by the Mapping Engine to connect between the external CIs and the RTSM CIs.

Reconciliation CI type. The type of CIs that represent reconciliation objects. These CIs must be stored in both the RTSM and in the external data repositories.

Mapping engine. A component that identifies relations between CIs from different data repositories that have a virtual relationship between them. The identification is performed by reconciling RTSM reconciliation objects and external CI reconciliation objects.

Mapping Engine

Federation Framework uses the Mapping Engine to calculate the federated TQL query. The Mapping Engine connects between CIs that are received from different data repositories and are connected by virtual relationships. The Mapping Engine also provides reconciliation data for the virtual relationship. One end of the virtual relationship must refer to the RTSM. This end is a `reconciliation` type. For the calculation of the two subgraphs, a virtual relationship can start from any end node.

Federated Adapter

The Federated adapter brings two kinds of data from external data repositories: external CI data and reconciliation objects that belong to external CIs.

- **External CI data.** The external data that does not exist in the RTSM. It is the target data of the external data repository.

- **Reconciliation object data.** The auxiliary data that is used by the federation framework to connect between RTSM CIs and external data. Each reconciliation object should refer to an External CI. The type of reconciliation object is the type (or subtype) of one of the virtual relationship ends from which data is retrieved. Reconciliation objects should fit the adapter received to reconciliation data. The reconciliation object can be one of three types:
`IdReconciliationObject`, `PropertyReconciliationObject`, or `TopologyReconciliationObject`.

In the `DataAdapter`-based interfaces (`DataAdapter`, `PopulateDataAdapter`, and `PopulateChangesDataAdapter`), the reconciliation is requested as part of the query definition.

Interactions between the Federation Framework, Server, Adapter, and Mapping Engine

The following diagrams illustrate the interactions between the Federation Framework, BSM Server, the adapter, and the Mapping Engine. The federated TQL query in the example diagrams has only one virtual relationship, so that only the BSM and one external data repository are involved in the federated TQL query.

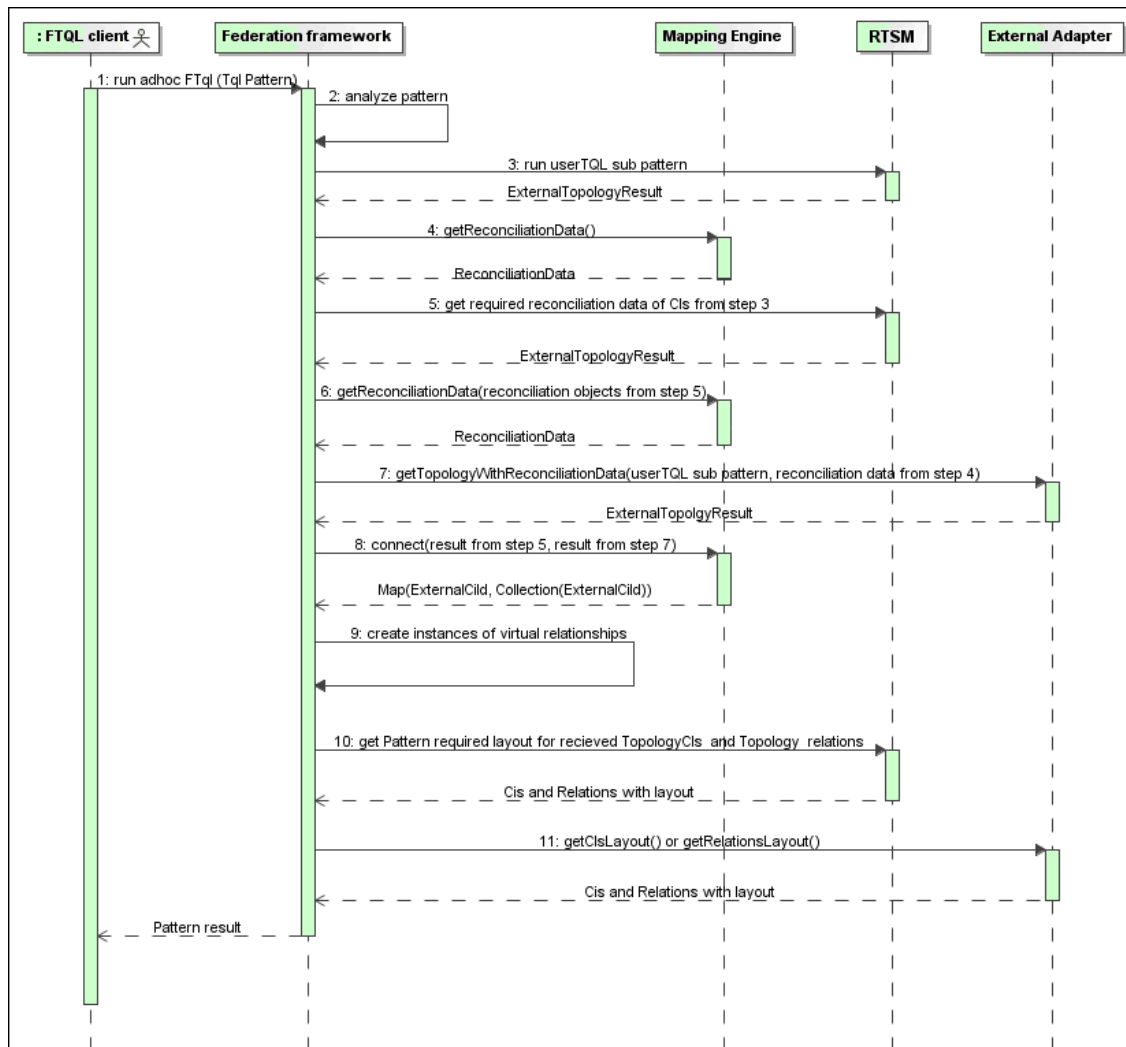
This section includes the following topics:

- ["Calculation Starts at the Server End" below](#)
- ["Calculation Starts at the External Adapter End" on page 188](#)
- ["Example of Federation Framework Flow for Federated TQL Queries" on page 189](#)

In the first diagram the calculation begins in the BSM and in the second diagram in the external adapter. Each step in the diagram includes references to the appropriate method call of the adapter or mapping engine interface.

Calculation Starts at the Server End

The following sequence diagram illustrates the interaction between the Federation Framework, BSM, the adapter, and the Mapping Engine. The federated TQL query in the example diagram has only one virtual relationship, so that only BSM and one external data repository are involved in the federated TQL query.

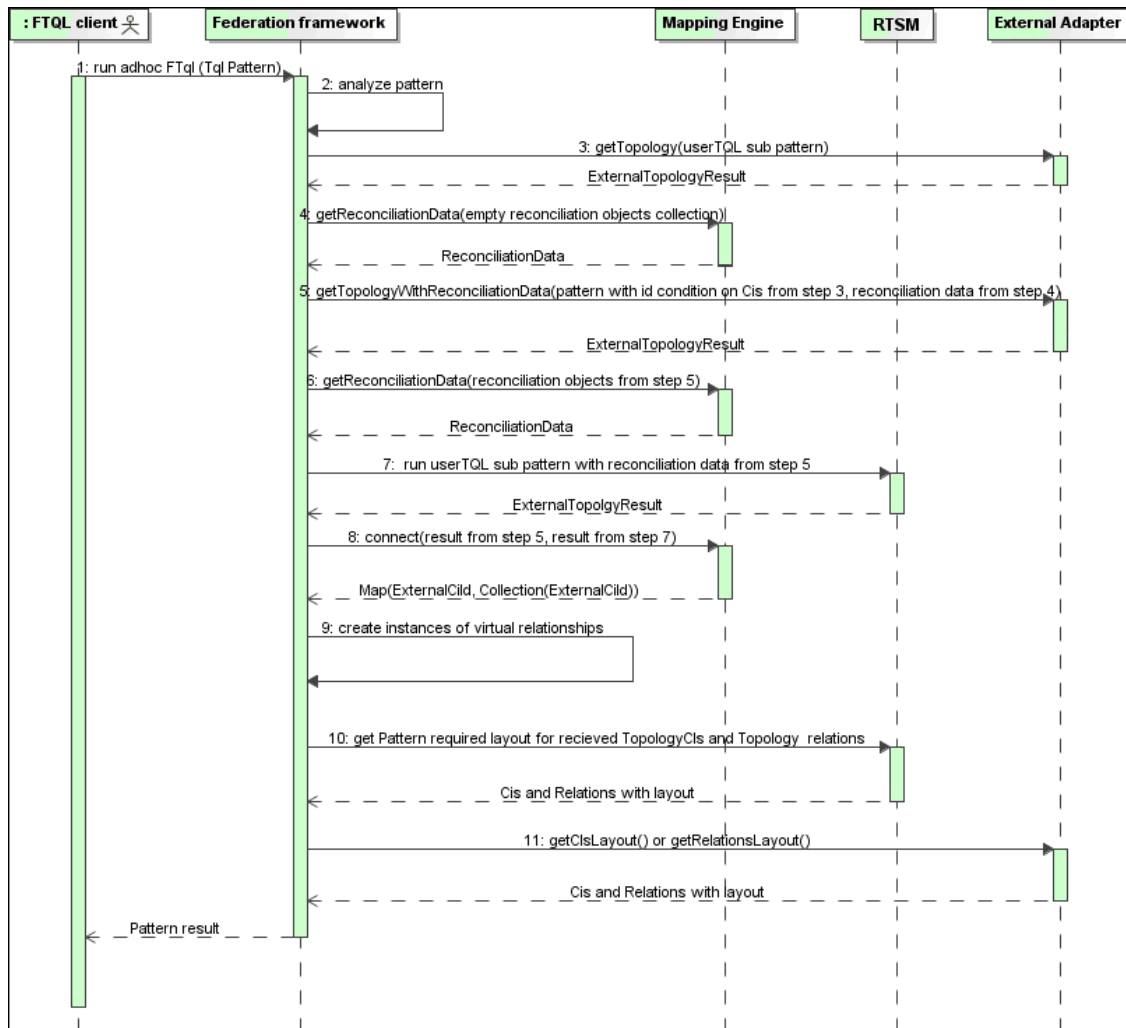


The numbers in this image are explained below:

Number	Explanation
1	The Federation Framework receives a call for a federated TQL calculation.
2	The Federation Framework analyzes the adapter, finds the virtual relationship, and divides the original TQL into two sub-adapters—one for BSM and one for the external data repository.
3	The Federation Framework requests the topology of the sub-TQL from BSM.

Number	Explanation
4	<p>After receiving the topology results, the Federation Framework calls the appropriate Mapping Engine for the current virtual relationship and requests reconciliation data. The <code>reconciliationObject</code> parameter is empty at this stage, that is, no condition is added to reconciliation data in this call. The returned reconciliation data defines which data is needed to match the reconciliation CIs in BSM to the external data repository. The reconciliation data can be one of the following types:</p> <ul style="list-style-type: none">• IdReconciliationData. CIs are reconciled according to their ID.• PropertyReconciliationData. CIs are reconciled according to the properties of one of the CIs.• TopologyReconciliationData. CIs are reconciled according to the topology (for example, to reconcile node CIs, the IP address of IP is required too).
5	<p>The Federation Framework requests reconciliation data for the CIs of the virtual relationship ends that were received in step "3" on the previous page from BSM.</p>
6	<p>The Federation Framework calls the Mapping Engine to retrieve the reconciliation data. In this state (by contrast with step "3" on the previous page), the Mapping Engine receives the reconciliation objects from step "5" above as parameters. The Mapping Engine translates the received reconciliation object to the condition on the reconciliation data.</p>
7	<p>The Federation Framework requests the topology of the sub-TQL from the external data repository. The external adapter receives the reconciliation data from step "6" above as a parameter.</p>
8	<p>The Federation Framework calls the Mapping Engine to connect between the received results. The <code>firstResult</code> parameter is the external topology result received from BSM in step "5" above and the <code>secondResult</code> parameter is the external topology result received from the External Adapter in step "7" above. The Mapping Engine returns a map where External CI ID from the first data repository (BSM in this case) is mapped to the External CI IDs from the second (external) data repository.</p>
9	<p>For each mapping, the Federation Framework creates a virtual relationship.</p>
10	<p>After the calculation of the federated TQL query results (only at the topology stage), the Federation Framework retrieves the original TQL layout for the resulting CIs and relationships from the appropriate data repositories.</p>

Calculation Starts at the External Adapter End



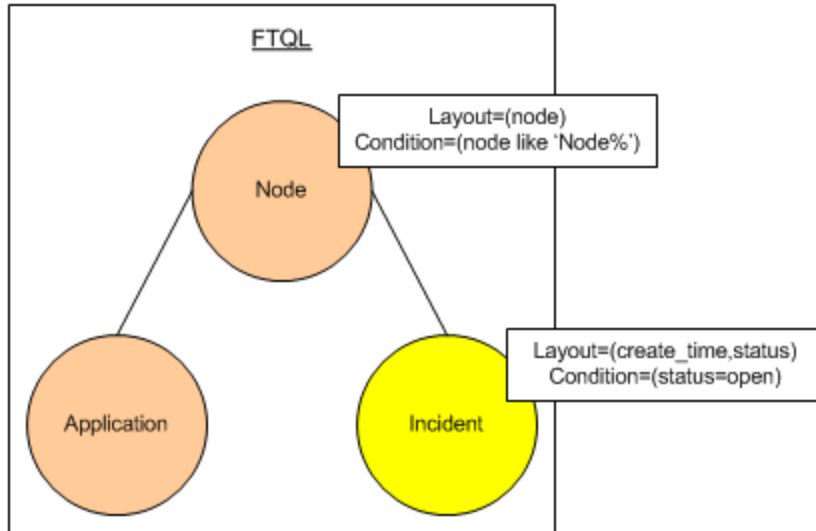
The numbers in this image are explained below:

Number	Explanation
1	The Federation Framework receives a call for an federated TQL calculation.
2	The Federation Framework analyzes the adapter, finds the virtual relationship, and divides the original TQL into two sub-adapters – one for BSM and one for the external data repository.
3	The Federation Framework requests the topology of the sub-TQL from the External Adapter. The returned <code>ExternalTopologyResult</code> is not supposed to contain any reconciliation object, since the reconciliation data is not part of the request.

Number	Explanation
4	<p>After receiving the topology results, the Federation Framework calls the appropriate Mapping Engine with the current virtual relationship and requests reconciliation data. The <code>reconciliationObjects</code> parameter is empty at this state, that is, no condition is added to the reconciliation data in this call. The returned reconciliation data defines what data is needed to match the reconciliation CIs in BSM to the external data repository. The reconciliation data can be one of three following types:</p> <ul style="list-style-type: none"> • IdReconciliationData. CIs are reconciled according to their ID. • PropertyReconciliationData. CIs are reconciled according to the properties of one of the CIs. • TopologyReconciliationData. CIs are reconciled according to the topology (for example, to reconcile node CIs, the IP address of IP is required too).
5	<p>The Federation Framework requests reconciliation objects for the CIs that were received in step 3 from the external data repository. The Federation Framework calls the getTopologyWithReconciliationData() method in the External Adapter, where the requested topology is a one-node topology with CIs received in step 3 as the ID condition and reconciliation data from step 4.</p>
6	<p>The Federation Framework calls the Mapping Engine to retrieve the reconciliation data. In this state (by contrast with step 3), the Mapping Engine receives the reconciliation objects from step 5 as parameters. The Mapping Engine translates the received reconciliation object to the condition on the reconciliation data.</p>
7	<p>The Federation Framework requests the topology of the sub-TQL with reconciliation data from step 6 from BSM.</p>
8	<p>The Federation Framework calls the Mapping Engine to connect between the received results. The <code>firstResult</code> parameter is the external topology result received from the External Adapter at step 5 and the <code>secondResult</code> parameter is the external topology result received from BSM at step 7. The Mapping Engine returns a map where the External CI ID from the first data repository (the external data repository in this case) is mapped to the External CI IDs from the second data repository (BSM).</p>
9	<p>For each mapping, the Federation Framework creates a virtual relationship.</p>
10	<p>After the calculation of the federated TQL query results (only at the topology stage), the Federation Framework retrieves the original TQL layout for the resulting CIs and relationships from the appropriate data repositories.</p>

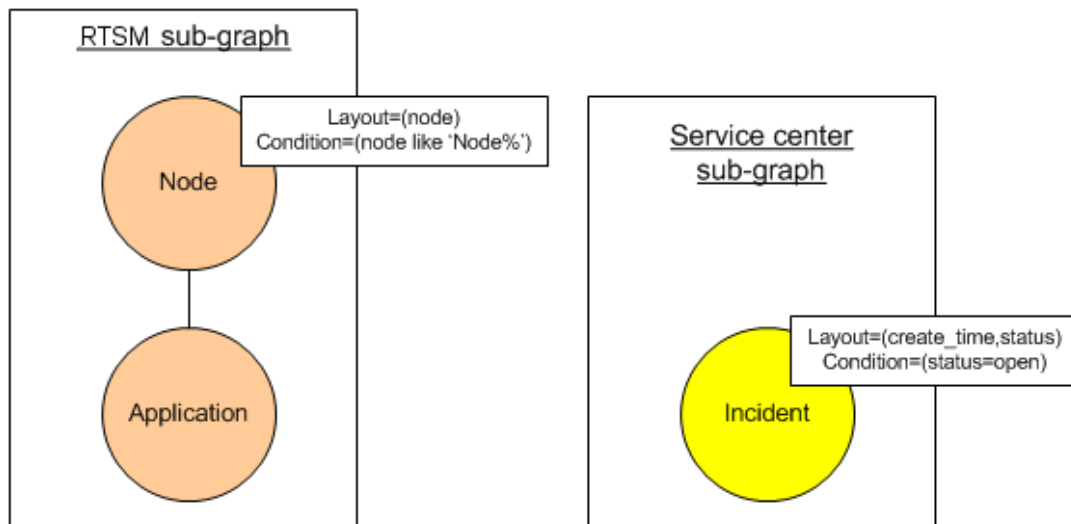
Example of Federation Framework Flow for Federated TQL Queries

This example explains how to view all open incidents on specific nodes. The ServiceCenter data repository is the external data repository. The node instances are stored in BSM, and the incident instances are stored in ServiceCenter. It is assumed that to connect the incident instances to the appropriate node, the `node` and `ip_address` properties of the host and IP are needed. These are reconciliation properties that identify the nodes from ServiceCenter in BSM.

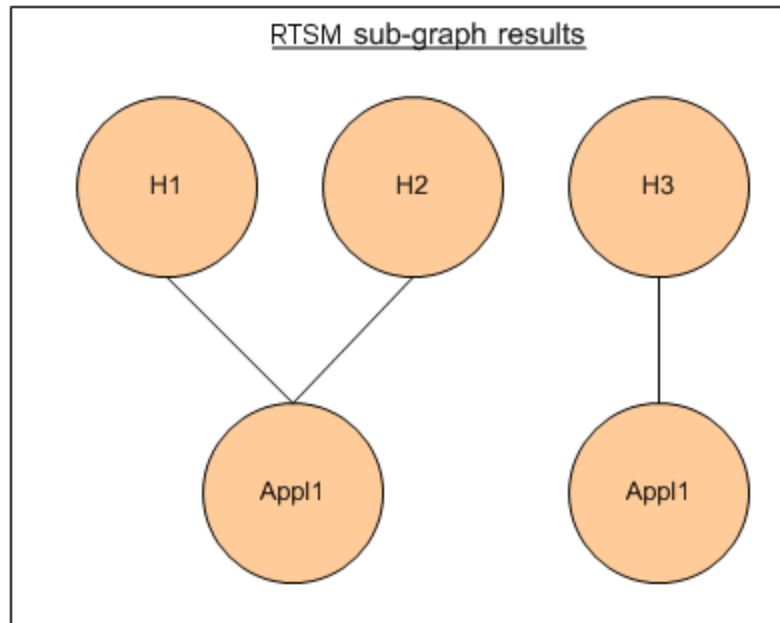


Note: For attribute federation, the adapter's **getTopology** method is called. The reconciliation data is adapted in the user TQL (in this case, the CI element).

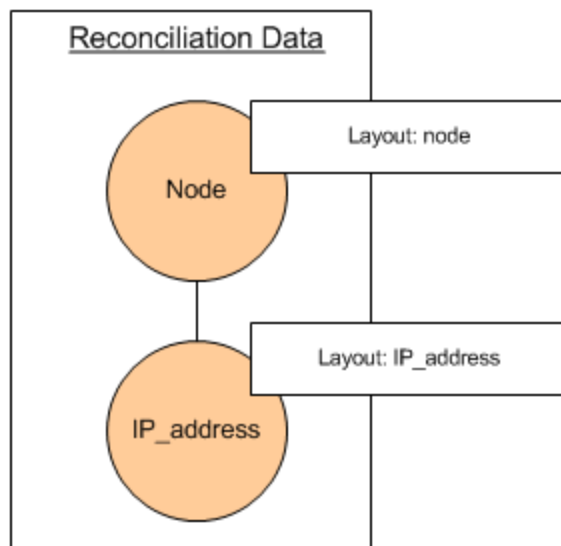
1. After analyzing the adapter, the Federation Framework recognizes the virtual relationship between **Node** and **Incident** and splits the federated TQL query into two subgraphs:



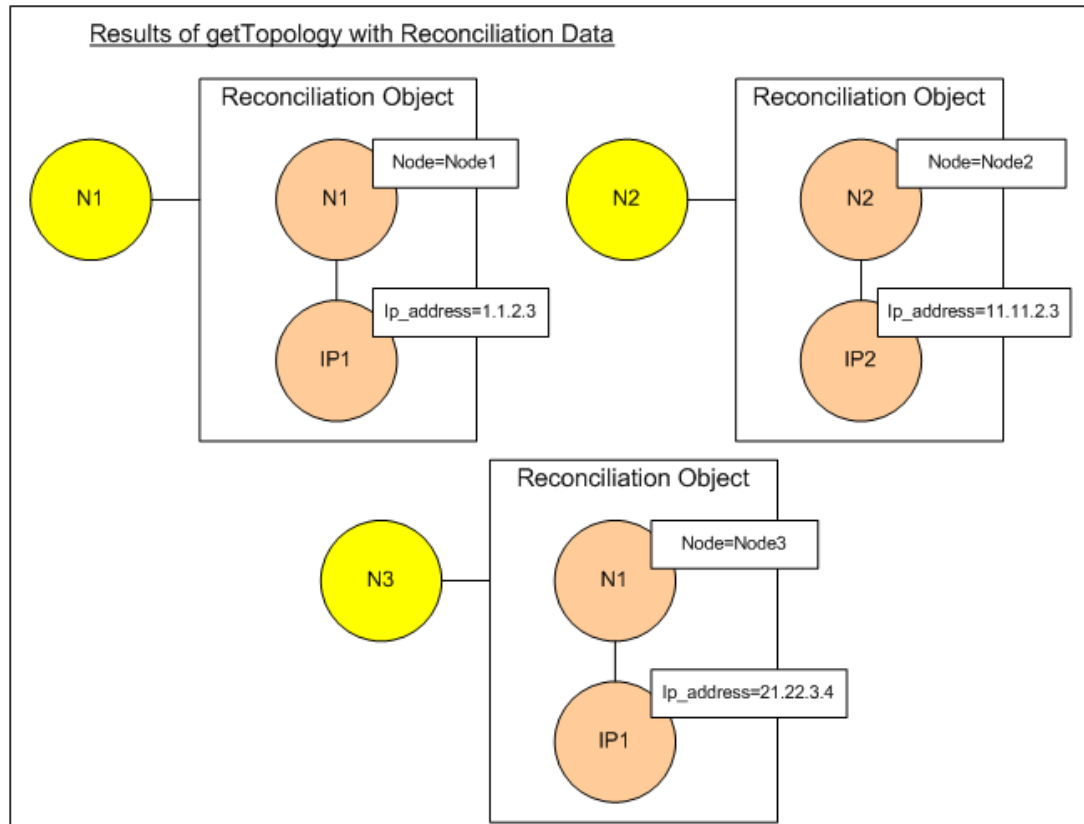
2. The Federation Framework runs the BSM subgraph to request the topology, and receives the following results:



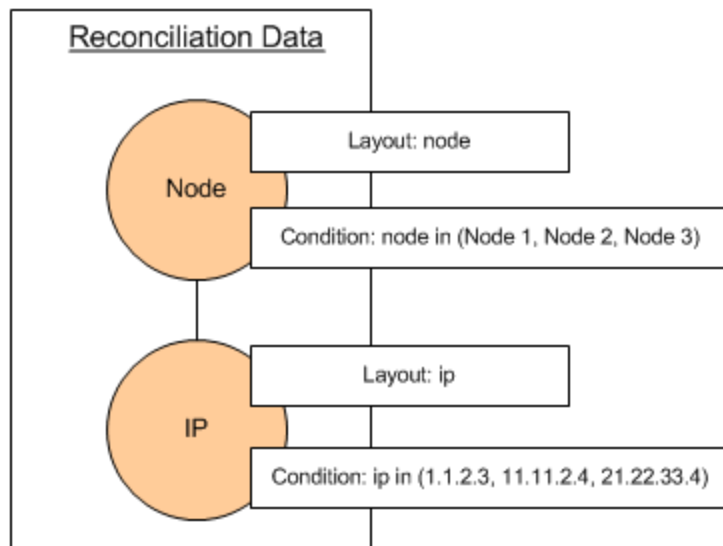
3. The Federation Framework requests, from the appropriate Mapping Engine, the reconciliation data for the first data repository (BSM) that contains the information to connect between received data from two data repositories. The reconciliation data in this case is:



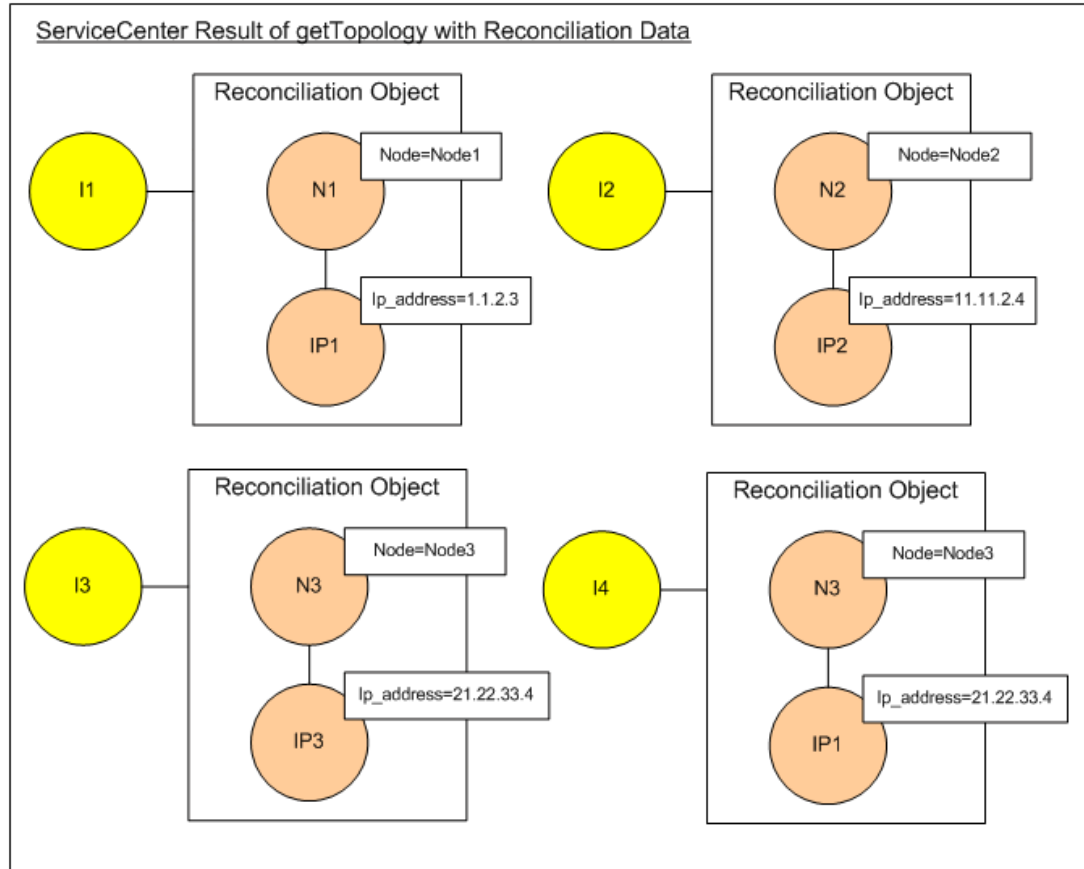
4. The Federation Framework creates a one-node topology query with the Node and ID conditions on it from the previous result (`node` in H1, H2, H3), and runs this query with the required reconciliation data on BSM. The result includes Node CIs that are relevant to the ID condition and the appropriate reconciliation object for each CI:



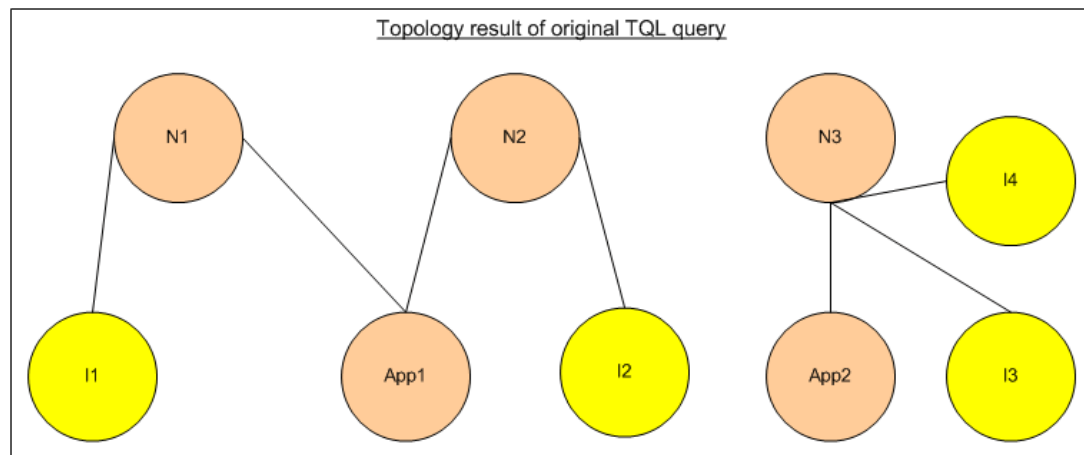
5. The reconciliation data for ServiceCenter should contain a condition for `node` and `ip` that is derived from the reconciliation objects received from BSM:



6. The Federation Framework runs the ServiceCenter subgraph with the reconciliation data to request the topology and appropriate reconciliation objects, and receives the following results:



7. The result after connection in Mapping Engine and creating virtual relationships is:



8. The Federation Framework requests the original TQL layout for received instances from BSM and ServiceCenter.

Federation Framework Flow for Population

This section includes the following topics:

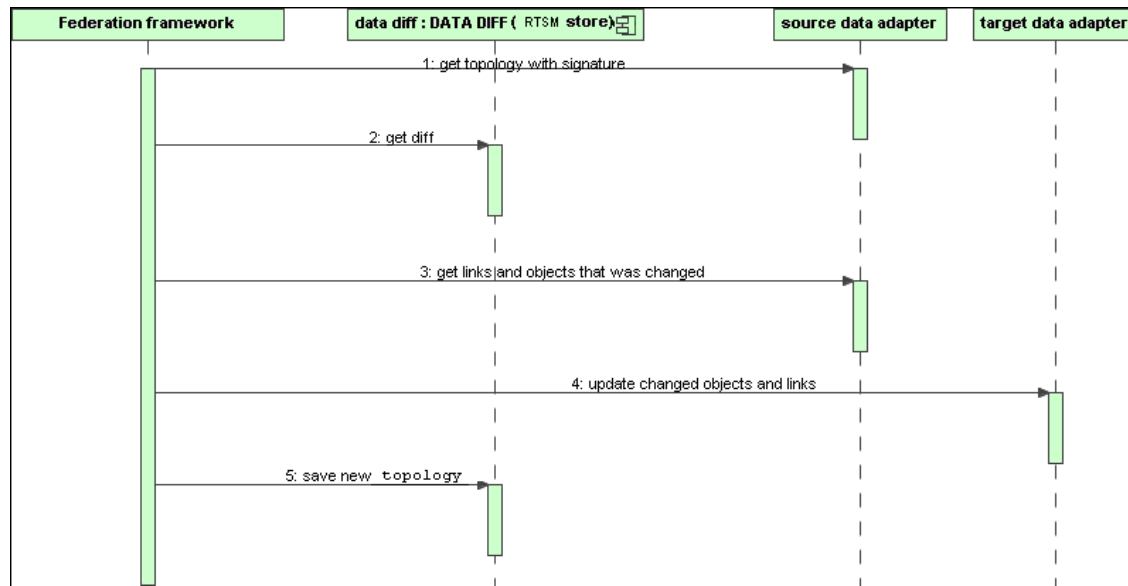
- ["Definitions and Terms" below](#)
- ["Flow Diagram" below](#)

Definitions and Terms

Signature. Denotes the state of properties in the CI. If changes are made to property values in a CI, the CI signature must also be changed. The CI signature helps to detect whether a CI has changed without retrieving and comparing all CI properties. Both the CI and the CI signature are provided by the appropriate adapter. The adapter is responsible for changing the CI signature when the CI properties are altered.

Flow Diagram

The following sequence diagram illustrates the interaction between the Federation Framework and the source and target adapters in a population flow:



1. The Federation Framework receives the topology for the query result from the source adapter. The adapter recognizes the query by its name and runs it on the external data repository. The topology result contains the ID and signature for each CI and relationship in the result. The ID is the logical ID that defines the CI as unique in the external data repository. The signature should be modified if the CI or relationship is modified.
2. The Federation Framework uses signatures to compare the newly received topology query results with the saved ones, and to determine which CIs have changed.
3. After the Federation Framework finds the CIs and relationships that have changed, it calls the source adapter with the IDs of the changed CIs and relationships as a parameter to retrieve their full layout.
4. The Federation Framework sends the update to the target adapter. The target adapter updates

the external data source with the received data.

5. After the update, the Federation Framework saves the last query result.

Adapter Interfaces

This section includes the following topics:

- ["Definitions and Terms" below](#)
- ["Adapter Interfaces for Federated TQL Queries" below](#)

Definitions and Terms

External Relation. The relation between two external CI types that are supported by the same adapter.

Adapter Interfaces for Federated TQL Queries

Use the appropriate adapter interface for each adapter, as follows.

- A **one Node topology interface** is used when the adapter does not support any external relations; that is, the adapter is never meant to receive a request with more than one external CI. All OneNode interfaces are created to simplify the workflow; for those cases where you need to use a more extensive query, use the `DataAdapter` interface.
- A **DataAdapter interface** is used to define adapters that support complex federated queries. The reconciliation request in these adapters is part of the single `QueryDefinition` parameter. These adapters may also be used for Population.
- **Pattern topology interface (Deprecated as of UCMDB 9.00)**

OneNode Interfaces

The following interfaces have different types of reconciliation data:

- **OneNodeTopologyIdReconciliationDataAdapter.** Use if the adapter supports a **single-node TQL** and the reconciliation between data repositories is calculated by the ID.
- **OneNodeTopologyPropertyReconciliationDataAdapter.** Use if the adapter supports a **single-node TQL** and the reconciliation between data repositories is done by the properties of one CI.
- **OneNodeTopologyDataAdapter.** Use if the adapter supports a **single-node TQL** and the reconciliation between data repositories is done by topology.

Data Adapter Interfaces

- **DataAdapter.** Use this adapter to support complex federated TQL queries. Allows the most diversity.
- **PopulateDataAdapter.** Use this adapter to support complex federated TQL queries and population flows. In a population flow, this adapter retrieves the entire data set, and lets the probe filter the difference since the last execution of the job.
- **PopulateChangesDataAdapter.** Use this adapter to support complex federated TQL queries and population flows. In a population flow, this adapter supports the retrieval of only the changes that occurred since the last execution of the job.

Note: When developing an adapter that may return large data sets of data, its important to allow chunking by implementing the ChunkGetter Interface. See the Java document of the specific adapter for more information.

Pattern Topology Interfaces (Deprecated as of UCMDB 9.00)

The following interfaces have different types of reconciliation data:

- **PatternTopologyIdReconciliationDataAdapter.** Use if the adapter supports a **complex TQL** and the reconciliation between data repositories is done by the ID.
- **PatternTopologyPropertyReconciliationDataAdapter.** Use if the adapter supports a **complex TQL** and the reconciliation between data repositories is done by single-node properties.
- **PatternTopologyDataAdapter.** Use if the adapter supports a **complex TQL** and the reconciliation between data repositories is done by topology.

Additional Interfaces

- **SortResultDataAdapter.** Use if you can sort the resulting CIs in the external data repository.
- **FunctionalLayoutDataAdapter.** Use if you can calculate the functional layout in the external data repository.

Adapter Interfaces for Synchronization

- **SourceDataAdapter.** Use for source adapters in population flows.
- **TargetDataAdapter.** Use for target adapters in data push flows.

Debug Adapter Resources

This task describes how to use the JMX console to create, view, and delete adapter state resources (any resources created using the resource manipulation methods in the `DataAdapterEnvironment` interface, which are saved in the RTSM database or the Probe database) for debugging and development purposes.

1. Launch the Web browser and enter the server address, as follows:

- For the RTSM server: `http://localhost:8080/jmx-console`
- For the Probe: `http://localhost:1977`

You may have to log in with a user name and password (the defaults are `sysadmin/sysadmin`).

2. To open the JMX MBEAN View page, do one of the following:

- On the RTSM server: click **UCMDB:service=FCMDB Adapter State Resource Services**
- On the Probe: click **type=AdapterStateResources**

3. Enter values in the operations that you want to use, and click **Invoke**.

Add an Adapter for a New External Data Source

This task explains how to define an adapter to support a new external data source.

This task includes the following steps:

- ["Prerequisites" below](#)
- ["Define Valid Relationships for Virtual Relationships" below](#)
- ["Define an Adapter Configuration" on the next page](#)
- ["Define Supported Classes" on page 203](#)
- ["Implement the Adapter" on page 204](#)
- ["Define Reconciliation Rules or Implement the Mapping Engine" on page 204](#)
- ["Add Jars Required for Implementation to the Class Path" on page 204](#)
- ["Deploy the Adapter" on page 204](#)
- ["Update the Adapter" on page 205](#)

1. Prerequisites

Model-supported adapter classes for CIs and relationships in the BSM Data Model. As an adapter developer, you should:

- have knowledge of the hierarchy of the BSM CI types to understand how external CITs are related to the BSM CITs
- model the external CITs in the BSM class model
- add the definitions for new CI types and their relationships
- define valid relationships in the BSM class model for the valid relationships between adapter inner classes. (The CITs can be placed at any level of the BSM class model tree.)

Modeling should be the same regardless of federation type (on the fly or replication). For details on adding new CIT definitions to the BSM class model, see *Working with the CI Selector in the Modeling Guide*.

For the adapter to support federated attributes on CITs, add this CIT to the supported classes with supported attributes and the reconciliation rule for this CIT.

2. Define Valid Relationships for Virtual Relationships

Note: This section is relevant only for federation.

To retrieve federated CITs that are connected to local RTSM CITs, a valid link definition must exist between the two CITs in the RTSM.


- a. Create a valid links XML file that contains these links (if they do not already exist).
- b. Add the links XML file to the adapter package in the **validlinks** folder. For details, see "Package Manager" in the *RTSM Administration Guide*.

Example of Valid Relationship Definition:

In the following example, the relation of type `containment` between instances of type `node` to instances of type `myclass1` is a valid relationship definition.

```
<Valid-Links>
  <Valid-Link>
    <Class-Ref class-name="containment">
      <End1 class-name="node">
        <End2 class-name="myclass1">
          <Valid-Link-Qualifiers>
        </Valid-Link-Qualifiers>
      </End2>
    </End1>
  </Valid-Link>
</Valid-Links>
```

3. Define an Adapter Configuration

- a. Navigate to **Admin > RTSM Administration > Adapter Management**.
- b. Click the **Create new resource**  button.
- c. In the New adapter dialog box, select **Integration** and **Java Adapter**.
- d. Right-click on the adapter that you created and select **Edit Adapter Source** from the shortcut menu.
- e. Edit the following XML tags:

```
<pattern xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
id="newAdapterIdName"
xsi:noNamespaceSchemaLocation="../../../Patterns.xsd"
description="Adapter Description" schemaVersion="9.0"
displayName="New Adapter Display Name">

<deletable>true</deletable>

<discoveredClasses>

<discoveredClass>link</discoveredClass>

<discoveredClass>object</discoveredClass>

</discoveredClasses>

<taskInfo
className="com.hp.ucmdb.discovery.probe.services.dynamic.core.
AdapterService">

<params
className="com.hp.ucmdb.discovery.probe.services.dynamic.core.
AdapterServiceParams" enableAging="true"
enableDebugging="false" enableRecording=
"false" autoDeleteOnErrors="success" recordResult="false"
maxThreads="1" patternType="java_adapter"
maxThreadRuntime="25200000">
```



```
<className>com.yourCompany.adapter.MyAdapter.MyAdapterClass
</className>

</params>

<destinationInfo
class-
Name="com.hp.ucmdb.discovery.probe.tasks.BaseDestinationData">

<!-- check -->

<destinationData name="adapterId"
description="">${ADAPTER.adapter_id}</destinationData>

<destinationData name="attributeValues"
description="">${SOURCE.attribute_values}</destinationData>

<destinationData name="credentialsId"
description="">${SOURCE.credentials_id}</destinationData>

<destinationData name="destinationId"
description="">${SOURCE.destination_id}</destinationData>

</destinationInfo>

<resultMechanism isEnabled="true">

<autoDeleteCITs isEnabled="true">

<CIT>link</CIT>

<CIT>object</CIT>

</autoDeleteCITs>

</resultMechanism>

</taskInfo>

<adapterInfo>

<adapter-capabilities>

<support-federated-query>

<!--<supported-classes/> <!--see the section about supported
classes-->

<topology>

<pattern-topology /> <!--or <one-node-topology> -->

</topology>

</support-federated-query>

<!--<support-replicatioin-data>

<source>

<changes-source/>

</source>
```

```
<target/>

</adapter-capabilities>

<default-mapping-engine />

<queries />

<removedAttributes />

<full-population-days-interval>-1</full-population-days-
interval>

</adapterInfo>

<inputClass>destination_config</inputClass>

<protocols />

<parameters>

<!--The description attribute may be written in simple text or
HTML.-->

<!--The host attribute is treated as a special case by UCMDB-->

<!--and will automatically select the probe name (if possible)--
>

<!--according to this attribute's value.-->

<parameter name="credentialsId" description="Special type of
property, handled by UCMDB for credentials menu" type="integer"
display-name="Credentials ID" mandatory="true" order-index="12"
/>

<parameter name="host" description="The host name or IP address
of the remote machine" type="string" display-name="Hostname/IP"
mandatory="false" order-index="10" />

<parameter name="port" description="The remote machine's
connection port" type="integer" display-name="Port"
mandatory="false" order-index="11" />

</parameters>

<parameter name="myatt" description="is my att true?"
type="string" display-name="My Att" mandatory="false" order-
index="15" valid-values="True;False"/>True</parameters>

<collectDiscoveredByInfo>true</collectDiscoveredByInfo>

<integration isEnabled="true">

<category >My Category</category>

</integration>

<overrideDomain>${SOURCE.probe_name}</overrideDomain>

<inputTQL>
```

```

<resource:XmlResourceWrapper
  xmlns:resource="http://www.hp.com/ucmdb/1-0-
0/ResourceDefinition" xmlns:ns4="http://www.hp.com/ucmdb/1-0-
0/ViewDefinition" xmlns:tql="http://www.hp.com/ucmdb/1-0-
0/TopologyQueryLanguage">

  <resource xsi:type="tql:Query" group-id="2" priority="low" is-
live="true" owner="Input TQL" name="Input TQL">

    <tql:node class="adapter_config" id="-11" name="ADAPTER" />

    <tql:node class="destination_config" id="-10" name="SOURCE" />

    <tql:link to="ADAPTER" from="SOURCE" class="fcmdb_conf_
aggregation" id="-12" name="fcmdb_conf_aggregation" />

  </resource>

</resource:XmlResourceWrapper>

</inputTQL>

<permissions />

</pattern>

```

For details about the XML tags, see ["XML Configuration Tags and Properties "](#) on page 208.

4. Define Supported Classes

Define supported classed either the adapter code by implementing the *getSupportedClasses()* method, or by using the pattern XML file.

```

<supported-classes>
  <supported-class name="HistoryChange" is-derived="false" is-
reconciliation-supported="false" federation-not-
supported="false" is-id-reconciliation-supported="false">
    <supported-conditions>
      <attribute-operators attribute-name="change_create_
time">
        <operator>GREATER</operator>
        <operator>LESS</operator>
        <operator>GREATER_OR_EQUAL</operator>
        <operator>LESS_OR_EQUAL</operator>
        <operator>CHANGED_DURING</operator>
      </attribute-operators>
    </supported-conditions>
  </supported-class>

```

name	The name of the CI type
is-derived	Specifies whether this definition includes all inheriting children

is-reconciliation-supported	Specifies whether this class is used for reconciliation
is-id-reconciliation-supported	Specifies whether this class is used for id-reconciliation
federation-not-supported	Specifies whether this CIT should not be allowed for federation (blocking certain CITs, for example, a CIT defined solely for federation)
<supported-conditions>	Specifies the supported conditions on each attribute

5. Implement the Adapter

Select the correct adapter implementation class according to its defined capabilities. The adapter implementation class implements the appropriate interfaces according to defined capabilities.

6. Define Reconciliation Rules or Implement the Mapping Engine

If your adapter supports federated TQL queries, you have three options for defining your Mapping Engine:

- Use the default RTSM 9.x default mapping engine, which uses the RTSM's internal reconciliation rules for mapping. To use it, leave the **<default-mapping-engine>** XML tag empty.

For details, see ["The reconciliation_types.txt file" on page 152.](#)

- Use the RTSM 8.x mapping engine. To do this, use the following XML Tag: **<default-mapping-engine>com.hp.ucmdb.federation.mappingEngine.AdapterMappingEngine</default-mapping-engine>**

For details, see ["The reconciliation_rules.txt File \(for backwards compatibility\)" on page 152.](#)

- Write your own mapping engine by implementing the mapping engine interface and placing the JAR with the rest of the adapter code. To do this, use the following XML tag: **<default-mapping-engine>com.yourcompany.map.MyMappingEngine</default-mapping-engine>**

7. Add Jars Required for Implementation to the Class Path

To implement your classes, add the **federation_api.jar** file to your code editor class path.

8. Deploy the Adapter

Deploy the adapter package. For general details on deploying a package, see "Package Manager" in the *RTSM Administration Guide*.

The package should contain the following entities:

- New CIT definition (optional):
- Used only if the adapter supports new CI types that do not yet exist in BSM.
- The new CIT definitions are located in the `class` folder in the package.
- New data type definition (optional):
- Used only if the new CITs require new data types.
- The new data type definitions are located in the `typedef` folder in the package.
- New valid relationships definition (optional):
- Used only if the adapter supports the federated TQL.
- The new valid relationships definitions are located in the `validlinks` folder in the package.
- The pattern configuration XML file should be located in the `discoveryPatterns` folder in the package.
- **Descriptor.** Defines the package definitions.
- Place your compiled classes (normally a jar file) in the package under the **adapterCode\<adapter id>** folder.

Note: The `adapter id` folder name has the same value as in the adapter configuration.

- If you create your own configuration file, you should place the file in the package under the **adapterCode\<adapter id>** folder.

9. Update the Adapter

Changes to any of the adapter's non-binary files may be made in the Adapter Management module. Making changes to configuration files in the Adapter management module causes the adapter to reload with the new configurations.

Updates may also be made by editing the files in the package (both binary and non-binary files), and then redeploying the package by using the Package Manager. For details, see *Deploy a Package* in the *RTSM Administration Guide*.

Implement the Mapping Engine

The configuration of the mapping engine depends on which mapping engine you are using.

This task includes the following steps:

- "Configure the `reconciliation_types.txt` File (for the BSM 9.x default mapping engine)" below
- "Configure the `reconciliation_rules.txt` File (for the BSM 8.x mapping engine)" below

1. Configure the `reconciliation_types.txt` File (for the BSM 9.x default mapping engine)

The file is used to define which CI types are used for reconciliation in the adapter.

Write each CI types used for reconciliation on a single line, as follows:

```
node
business_application
```

Place the file in the adapter package in the `adapterCode\<AdapterID>\META-INF\` folder. To support ID reconciliation (reconciliation based on ID mapping between the RTSM ID in the RTSM to a value on the remote database), you should map a special RTSM attribute called `cmdb_id` to a column in the database of either the string (char, varchar) or byte[] (raw/bytes) type.

2. Configure the `reconciliation_rules.txt` File (for the BSM 8.x mapping engine)

This file is used to configure the reconciliation rules. Each row in the file represents a rule. For example:

```
reconciliation_type[node] expression[^node.name OR ip_address.name]
end1_type[node] end2_type[ip_address] link_type[containment]
```

The **reconciliation_type** parameter is filled with the type of CI on which the reconciliation is performed (the BSM class name that is connected to the federated class in the TQL).

The **expression** parameter is the logic that decides whether two reconciliation objects are equal (one reconciliation object from the BSM side and the other from the Federated adapter side).

The expression is composed of ORs and ANDs.

The convention regarding attributes names in the expression part is **[className].[attributeName]**. For example, the attribute `ip_address` in the `ip` class is written `ip.ip_address`.

You can define ordered matches. The ordered match checks the first OR sub expression. If two reconciliation objects have the value on the attributes of the sub expression and it returns that false (the reconciliation objects are not equal) then the second OR sub expression is not compared.

For an ordered match, use **ordered expression** instead of **expression**.

The circumflex sign (^) is used to ignore case during comparisons.

The other parameters (**end1_type**, **end2_type**, and **link_type**) are used only if the reconciliation data contains two nodes and not just the node of the reconciliation type (the topological reconciliation data). In this case, the reconciliation data is **end1_type -(link_type)> end2_type**.

There is no need to add the relevant layout as it is retrieved from the expression.

To perform reconciliation by BSM ID, use **cmdb_id** as the attribute name in expression.

Place the file in the adapter package in the **adapterCode\<AdapterID>\META-INF** folder.

Examples:

- You can add a reconciliation rule for a node CIT only. This is because only node CITs have valid relationships with external CITs. For example, a node CI in the RTSM is matched to a node CI in ServiceCenter through the `node.name` attribute or through the `ip_address.name` attribute.
- The reconciliation rule in this case is a topology rule and the expression is ordered. The rule performs the following checks on the CIs under comparison:
 - If the `node.name` attribute is equal, the rule matches the nodes.
 - If the `node.name` attribute is not equal, the rule does not match the nodes.
 - If the `node.name` attribute is null in one of the compared CIs, the rule checks the `ip_address.name` attribute. If the `ip_address.name` attribute is equal, the rule matches the nodes.

Create a Sample Adapter

This example illustrates how to create a sample adapter. This task includes the following steps:

- "Select Adapter Logic" below
- "Load the Project" below

1. Select Adapter Logic

When you implement an adapter, you must choose how to handle the condition logic in the implementation (property conditions, ID conditions, reconciliation conditions, and link conditions).

- a. Retrieve the entire data into the adapter memory and let it select or filter the needed CI Instances.
- b. Convert all the conditions into the data source language and let it filter and select the data. For example:
 - Convert the condition into a SQL query.
 - Convert the condition into a Java API filter object.
- c. Filter some of the data on the remote service, and have the adapter select and filter the remainder.

In the MyAdapter example, the logic in option *a* is used.

2. Load the Project

Copy the files from the

<HP BSM root directory\odb\tools\adapter-dev-kit folder and follow the instructions in the readme files.

Note: If you use an adapter with large data sets, you may need to use caching and indexing to improve performance for Federation.

Online javadocs documentation is available at:

C:\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\DBAdapterFramework_JavaAPI\index.html

XML Configuration Tags and Properties

id="newAdapterIdName"	Defines the adapter's real name. Used for logs and folder lookups
displayName="New Adapter Display Name"	Defines the adapter's display name, as it appears in the UI.
<className>...</className>	Defines the adapter's interface implementing the Java class.

<category>My Category</category>		Defines the adapter's category.
<parameters>		Defines the properties for the configuration that are available in the UI when setting up a new integration point.
	name	The name of the property (used mostly by code)
	description	The display hint of the property
	type	String or integer (use valid values with string for Boolean).
	display-name	The name of the property in the UI.
	mandatory	Specifies whether this configuration property is mandatory for the user.
	order-index	The placing order of the property (small = up)
	valid-values	A list of possible valid values separated by `;` characters (for example, valid-values="Oracle;SQLServer;MySQL" or valid-values="True;False").
<adapterInfo>		Contains the definition of the adapter's static settings and capabilities.
	<support-federated-query>	Defines this adapter as capable of federation.
	<one-node-topology>	The ability to federated queries with one federated query node.
	<pattern-topology>	The ability to federate complex queries.
	<support-replication-data>	Defines the capability to run data push and population flows.
	<source>	This adapter may be used for population flows.
	<changes-source>	This adapter may be used for population changes flows.
	<target>	This adapter may be used for data push flows.
	<default-mapping-engine>	Allows definition of a mapping engine for the adapter (by default, the adapter uses the default mapping engine). For any other mapping engine, enter the implementing class name of the mapping engine (for the BSM 8.x mapping engine use: com.hp.ucmdb.federation.mappingEngine.AdapterMappingEngine)

	<removedAttributes>	Forces the removal of specific attributes from the result.
	<full-population-days-interval>	Specifies when to execute a full population job instead of a differential job (every `x` days). Uses the aging mechanism together with the changes flow.

Chapter 7

Developing Push Adapters

This chapter includes:

- Developing Push Adapters Overview 212
- Differential Synchronization 213
- Prepare the Mapping Files 214
- Write Jython Scripts 217
- Support Differential Synchronization 221
- Build an Adapter Package 223
- Mapping File Schema 225
- Mapping Results Schema 237

Developing Push Adapters Overview

The Generic Push Adapter provides a platform that enables rapid development of integrations that push BSM 9.x data to external data repositories (databases and third-party applications). Developing a custom integration based on Generic Push Adapter requires:

- An XML mapping file between the BSM CI link types and the external data items.
- A Jython script to push the data items into the external data repository.

Differential Synchronization

For the Push adapter to support differential synchronization, the **DiscoveryMain** function must return an object implementing the **DataPushResults** interface, which contains the mappings between the IDs that the Jython script receives from the XML and the IDs that the Jython script creates on the remote machine. The latter IDs are of the type **ExternalId**.

The **ExternalIdUtil.restoreExternal** command, which receives the ID of the CI in the CMDB as a parameter, restores the external ID from the ID of the CI in the CMDB. This command can be used, for example, while performing differential synchronization, and a link is received where one of its ends is not in the bulk (it was already synchronized).

If the **DiscoveryMain** method in the Jython script on which the Push adapter is based returns an empty **ObjectStateHolderVector** instance, the adapter will not support differential synchronization. This means that even when a differential synchronization job is run, in actuality, a full synchronization is being performed. Therefore, no data can be updated or removed on the remote system, since all data is added to the RTSM during each synchronization.

Prepare the Mapping Files

Note: You can retrieve all of the CIs and relationships as they are in the CMDB without mapping by not creating the **mappings.xml** file. This will return all of the CIs and relationships with all of their attributes.

There are two different ways to prepare mapping files:

- You can prepare a single, global mapping file.
All mappings are placed in a single file named **mappings.xml**.
- You can prepare a separate file for each push query.
Each mapping file is called **<query name>.xml**.

For details, see ["Mapping File Schema" on page 225](#).

This task includes the following steps:

- ["Create the Mapping File" below](#)
- ["Map CIs" below](#)
- ["Map Links " on the next page](#)

1. Create the Mapping File

The mapping file structure is created as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<integration>
  <info>
    <source name="UCMDB" versions="9.x" vendor="HP" >
      <!-- for example: -->
      <target name="Oracle" versions="11g" vendor="Oracle" >
    </info>
    <targetcis>
      <!-- CI Mappings --->
    </targetcis>
    <targetrelations>
      <!-- Link Mappings --->
    </targetrelations>
  </integration>
```

2. Map CIs

There are two ways to map RTSM CI types:

- Map a CI type so that CIs of that type and all inherited types are mapped in the same way:

```
<source_ci_type_tree name="node" mode="update_else_insert">
  <apioutputseq>1</apioutputseq>
  <target_ci_type name="host">
    <targetprimarykey>
```

```

        <pkey>name</pkey>
      </targetprimarykey>
      <target_attribute name=" name" datatype="STRING">
        <map type="direct" source_attribute="name" >
      </target_attribute>
      <!-- more target attributes --->
    </target_ci_type>
  </source_ci_type_tree>

```

- Map a CI type so that only CIs of that type will be processed. CIs of inherited types will not be processed unless their type is also mapped (in one of the two ways):

```

<source_ci_type name="node" mode="update_else_insert">
  <apioutputseq>1</apioutputseq>
  <target_ci_type name="host">
    <targetprimarykey>
      <pkey>name</pkey>
    </targetprimarykey>
    <target_attribute name=" name" datatype="STRING">
      <map type="direct" source_attribute="name" >
    </target_attribute>
    <!-- more target attributes --->
  </target_ci_type>
</source_ci_type>

```

A CI type which is mapped indirectly (one of its ancestors is mapped using **source_ci_type_tree**), can also override its parent's map by having it appear in its own **source_ci_type_tree** or **source_ci_type**.

It is recommended to use **source_ci_type_tree** wherever possible. Otherwise, resulting CIs of a CI type that do not appear in the mapping files will not be transferred to the Jython script.

3. Map Links

There are two ways to map links:

- Map a link that will also map all of the link types that inherit from that specific link:

```

<source_link_type_tree name="dependency" target_link_
type="dependency" mode="update_else_insert" source_ci_type_
end1="webservice" source_ci_type_end2="sap_gateway">
  <target_ci_type_end1 name="webservice" >
  <target_ci_type_end2 name="sap_gateway" >
    <target_attribute name="name" datatype="STRING">
      <map type="direct" source_attribute="name" >
    </target_attribute>
  </source_link_type_tree>

```

- Map a link that will also map only that specific link type and not the link types which inherit from it:

```

<link source_link_type="dependency" target_link_type="dependency"
mode="update_else_insert" source_ci_type_end1="webservice"
source_ci_type_end2="sap_gateway">
  <target_ci_type_end1 name="webservice" >

```

```
        <target_ci_type_end2 name="sap_gateway" >
        <target_attribute name="name" datatype="STRING">
            <map type="direct" source_attribute="name" >
        </target_attribute>
    </link>
```


Write Jython Scripts

The mapping script is a regular Jython script, and should follow the rules for Jython scripts. For details, see ["Developing Jython Adapters" on page 51](#).

The script should contain the **DiscoveryMain** function, which may return either an empty **OSHVResult** or a **DataPushResults** instance upon success.

To report any failure, the script should raise an exception, for example:

```
raise Exception('Failed to insert to remote UCMDB using  
TopologyUpdateService. See log of the remote UCMDB')
```

In the **DiscoveryMain** function, the data items to be pushed to or deleted from the external application can be obtained as follows:

```
# get add/update/delete result objects (in XML format) from the  
Framework  
addResult = Framework.getTriggerCIData('addResult')  
updateResult = Framework.getTriggerCIData('updateResult')  
deleteResult = Framework.getTriggerCIData('deleteResult')
```

The client object to the external application can be obtained as follows:

```
oracleClient = Framework.createClient()
```

This client object automatically uses the credentials ID, host name and port number passed by the adapter through the Framework.

If you need to use the connection parameters that you defined for the adapter (for details, see the step ["Edit the discoveryPatterns\push_adapter.xml file."](#) in ["Build an Adapter Package" on page 223](#)), use the following code:

```
propValue = str(Framework.getDestinationAttribute('<Connection  
Property Name'))
```

For example:

```
serverName = Framework.getDestinationAttribute('ip_address')
```

This section also includes:

- ["Working with the Mapping's Results" below](#)
- ["Handling Test Connection in the Script" on page 220](#)

Working with the Mapping's Results

The Generic Push Adapter creates XML strings that describe the data to be added, updated, or deleted from the target system. The Jython script needs to analyze this XML, and then performs the add, update, or delete operation on the target.

In the XML of the add operation that the Jython script receives, the `mamId` attribute for the objects and links is always the BSM identifier of the original object or link before its type, attribute or other information was changed to the schema of the remote system.

In the XML of the update or remove operations, the `mamId` attribute of each object or link contains the string representation of the same `ExternalId` that was returned from the Jython script from the previous synchronization.

In the XML, the `id` attribute of a CI holds the `cmdbId` as an external id or the `ExternalId` of that CI if the CI got an `ExternalId` one when the CI was sent to the script. The `end1Id` and `end2Id` fields of the link hold for each of the link's ends the `cmdbId` as an external id or the `ExternalId` of that link's end if the CI at the link's end got an `ExternalId` when it was sent to the script.

When processing the CIs in the Jython script, the return value of the script is a mapping between the CI's CMDB id and the given id (the id given to each CI in the script). If a CI is pushed for the first time, the id that is in the XML of that CI is the CMDB id. If a CI is not pushed for the first time, the CI's id is the same id that was given to that CI in the script when it was first pushed.

The id is retrieved from the CI XML script as follows:

1. From the CI Element in the XML, retrieve the id from the id attribute. For example: `id = objectElement.getAttributeValue('id')`.
2. After retrieving the id from the XML, restore the id from the attribute (string). For example: `objectId = CmdbObjectID.Factory.restoreObjectID(id)`.
3. Check if the `objectId` received in the previous step is the CMDB id. You can do this by checking if the `objectId` has the new id that is given to it by the script. If it does, the returned id is not the CMDB id. For example:
`newId = objectId.getPropertyValue(<the name of the id attribute which is given by the script>).`

If `newId` is null, then the id that was returned in the XML is a CMDB id.

4. If the id is a CMDB id (that is, `newId` is null), perform the following (if the id is not a CMDB id, go to step 5):
 - a. Create a property for that CI that holds the new id. For example: `propArray = [TypesFactory.createProperty('<the name of the id attribute which is given by the script>', '<new id>')]`.
 - b. Create an `externalId` to that CI. For example:
`cmdbId = extI.getPropertyValue('internal_id')`
`className = extI.getType()`
`externalId = ExternalIdFactory.createExternalCiId(className, propArray)`
 - c. Map the CMDB id to the new created `externalId` (and in the next step return that mapping to the adapter). For example: `objectMappings.put(cmdbId, externalId)`
 - d. When all of the CIs and links are mapped:
`updateResult = DataPushResultsFactory.createDataPushResults(objectMappings, linkMappings);`
`return updateResult`
5. If the id is the new id (that is, `newId` is not null), then the `externalId` is the `newId`.

Example of the XML result

```
<root>
```

```

<data>
  <objects>
    <Object mode="update_else_insert" name="UCMDB_UNIX"
operation="add" mamId="0c82f591bc3a584121b0b85efd90b174"
id="HiddenRmiDataSource%0Aunix%0A1%0Ainternal_
id%3DSTRING%3D0c82f591bc3a584121b0b85efd90b174%0A">
      <field name="NAME" key="false" datatype="char"
length="255">UNIX5</field>
      <field name="DATA_NOTE" key="false" datatype="char"
length="255"></field>
    </Object>
  </objects>
  <links>
    <link targetRelationshipClass="TALK" targetParent="unix"
targetChild="unix" operation="add" mode="update_else_insert"
mamId="265e985c6ec51a8543f461b30fa58f81"
id="end1Id%5BHiddenRmiDataSource%0Aunix%0A1%0Ainternal_
id%3DSTRING%3D41372a1cbcaba27b214b84a2ec9eb535%0A%5D%0Aend2Id%
5BHiddenRmiDataSource%0Aunix%0A1%0Ainternal_
id%3DSTRING%3D0c82f591bc3a584121b0b85efd90b174%0A%5D%0AHiddenRmi
DataSource%0Atalk%0A1%0Ainternal_
id%3DSTRING%3D265e985c6ec51a8543f461b30fa58f81%0A">
      <field
name="DiscoveryID1">41372a1cbcaba27b214b84a2ec9eb535</field>
      <field
name="DiscoveryID2">0c82f591bc3a584121b0b85efd90b174</field>
      <field name="end1Id">HiddenRmiDataSource%0Aunix%0A1%0Ainternal_
id%3DSTRING%3D41372a1cbcaba27b214b84a2ec9eb535%0A</field>
      <field name="end2Id">HiddenRmiDataSource%0Aunix%0A1%0Ainternal_
id%3DSTRING%3D0c82f591bc3a584121b0b85efd90b174%0A</field>
      <field name="NAME" key="false" datatype="char"
length="255">TALK4</field>
      <field name="DATA_NOTE" key="false" datatype="char"
length="255"></field>
    </link>
  </links>
</data>
</root>

```

Note: In case `datatype="BYTE"`, the returned result's value is a **String** that is generated as: `new String([the byte array attribute])`. The `byte[]` object can be reconstructed by: `<the received String>.getBytes()`. There might be difference in the default locale between the server and the probe. In this case, the reconstruction is according to the server's default locale.

Handling Test Connection in the Script

A Jython script can be invoked to test the connection with an external application. In this case, the `testConnection` destination attribute will be `true`. This attribute can be obtained from the Framework as follows:

```
testConnection = Framework.getTriggerCIData('testConnection')
```

When run in test connection mode, a script should raise an exception if a connection to the external application cannot be established. Otherwise, if the connection is successful, the **DiscoveryMain** function should return an empty **OSHVResult**.

Support Differential Synchronization

Important: If you are implementing differential synchronization on an existing adapter that was created in version 9.00 or 9.01, you must use the push-adapter.zip file from version 9.02 or later to recreate your adapter package. For details, see ["Build an Adapter Package" on page 223](#).

This task enables the Push adapter to perform differential synchronization. For details, see ["Differential Synchronization" on page 213](#).

The Jython script returns the **DataPushResults** object which contains two Java maps - one for object ID mappings (keys and values are ExternalCiid type objects) and one for link IDs (keys and values are ExternalRelationId type objects).

- Add the following **from** statements to your Jython script:

```
from com.hp.ucmdb.federationspi.data.query.types import
ExternalIdFactory

from com.hp.ucmdb.adapters.push import DataPushResults

from com.hp.ucmdb.adapters.push import DataPushResultsFactory

from com.mercury.topaz.cmdb.server.fcmbd.spi.data.query.types import
ExternalIdUtil
```

- Use the **DataPushResultsFactory** factory class to obtain the **DataPushResults** object from the **DiscoveryMain** function.

```
# Create the UpdateResult object

updateResult = DataPushResultsFactory.createDataPushResults
(objectMappings, linkMappings);
```

- Use the following commands to create Java maps for the **DataPushResults** object:

```
#Prepare the maps to store the mappings if IDs

objectMappings = HashMap()

linkMappings = HashMap()
```

- Use the **ExternalIdFactory** class to create the following ExternalId IDs:

- ExternalId for objects or links originating in a CMDB (for example, all of the CIs in an add operation are from the CMDB):

```
externalCiid = ExternalIdFactory.createExternalCmdbCiid(ciType,
ciIDAsString)

externalRelationId =
ExternalIdFactory.createExternalCmdbRelationId(linkType,
end1ExternalCiid,
end2ExternalCiid, linkIDAsString)
```

- ExternalId for objects or links not originating in a CMDB (usually, every update and remove operation contains such objects):

```
myIDField = TypesFactory.createProperty("systemID", "1")  
  
myExternalId = ExternalIdFactory.createExternalCiId(type,  
myIDField)
```

Note: If the Jython script updated existing information and the ID of the object (or link) changes, you must return a mapping between the previous external ID and the new one.

- Use the **restoreCmdbCiIDString** or **restoreCmdbRelationIDString** methods from the **ExternalIdFactory** class to retrieve the UCMDB ID string from an External ID of an object or link that originated in the UCMDB.
- Use the **restoreExternalCiid** and **restoreExternalRelationId** methods from the **ExternalIdUtil** class to restore the **ExternalId** object from the `mamId` attribute value of the XML of the update or remove operations.

Note: **ExternalId** objects are actually an array of properties. This means that you can use an **ExternalId** object to store any information you may need that will identify the data on the remote system.

Build an Adapter Package

1. Extract the content of
<HP BSM root directory\odb\content\adapters\push-adapter.zip into a temporary folder. In the adapter package, the **sql_queries** file located in **adapterCode > PushAdapter > sqlTablesCreation**, contains the queries needed to create tables in a new schema in Oracle for testing the adapter. The tables correspond to the **adapterCode\<adapter ID>\mappings\mappings.xml** file.

Note: The **sql_queries** file is not needed for the adapter. It is only an example.

2. Edit the **discoveryPatterns\push_adapter.xml** file.
 - a. Modify the **<pattern>** tag with a new id and display label. Replace:

```
<pattern id="PushAdapter"
xsi:noNamespaceSchemaLocation="../../../Patterns.xsd"
description="Discovery Pattern Description" schemaVersion="9.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

with:

```
<pattern id="MyPushAdapter" displayLabel="My Push Adapter"
xsi:noNamespaceSchemaLocation="../../../Patterns.xsd"
description="Discovery Pattern Description" schemaVersion="9.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```
 - b. Update the parameter list, so that the list of parameters reflects the required connection attributes. Do not remove the **probeName** attribute.
3. Rename the **adapterCode\PushAdapter** folder with the adapter ID used in the previous step (for example, **adapterCode\MyPushAdapter**).
4. In the **discoveryScript** file, there is a **script pushScript.py** script which inserts the CIs and links to an external Oracle database. Replace **discoveryScripts\pushScript.py** with the script you wrote (for details, see ["Write Jython Scripts" on page 217](#)). If you rename the script, the **jythonScript.name** property in **adapterCode\<adapter ID>\push.properties** should be updated accordingly.
5. The **adapterCode\<adapter ID>\mappings\mappings.xml** file, located in **adapterCode\<adapter ID>\mappings**, is a sample mapping file which contains a mapping of the:
 - Node CI type with all the CI types that inherit from it
 - UNIX CI type without the CI types that inherit from it
 - Dependency link with all of the link types that inherit from it
 - Talk link type without the inherited link types that inherit from it

This mapping example corresponds to the example of the tables created in ORACLE in the **sql_queries** file (see step 1).

Replace the **adapterCode\<adapter ID>\mappings\mappings.xml** file with the mapping files you prepared (for details, see ["Prepare the Mapping Files" on page 214](#)).

If you want to use a mapping file for each TQL method, assign the name of the corresponding TQL to each XML file, followed by **.xml**. In this case, the **mappings.xml** file will be used as a default, if no specific mapping file is found for the current TQL name. The name of the default mapping file can be modified by changing the `mappingFile.default` property in **adapterCode\<adapter ID>\push.properties**.

Mapping File Schema

Element Name and Path	Description	Attributes
integration	Defines the mapping contents of the file. Must be the outermost block in the file except for the beginning line and any comments.	
info (integration)	Defines information about the data repositories being integrated.	
source (integration > info)	Defines information about the source data repository.	<ol style="list-style-type: none">1. Name: type Description: Name of the source data repository. Is required?: Required Type: String2. Name: versions Description: Version(s) of the source data repositories. Is required?: Required Type: String3. Name: vendor Description: Vendor of the source data repository. Is required?: Required Type: String

Element Name and Path	Description	Attributes
target (integration > info)	Defines information about the target data repository.	<ol style="list-style-type: none"> 1. Name: type Description: Name of the source data repository. Is required?: Required Type: String 2. Name: versions Description: Version(s) of the source data repository. Is required?: Required Type: String 3. Name: vendor Description: Vendor of the source data repository. Is required?: Required Type: String
targetcis (integration)	Container element for all CIT mappings.	
source_ci_type_tree (integration > targetcis)	Defines a source CIT and all of the CI types which inherit from it.	<ol style="list-style-type: none"> 1. Name: name Description: Name of the source CIT. Is required?: Required Type: String 2. Name: mode Description: The type of update required for the current CI type. Is required?: Required Type: One of the following strings: <ol style="list-style-type: none"> a. insert: Use this only if the CI does not already exist. b. update: Use this only if the CI is known to exist. c. update_else_insert: If the CI exists, update it; otherwise, create a new CI. d. ignore: Do nothing with this CI type.

Element Name and Path	Description	Attributes
source_ci_type (integration > targetcis)	Defines a source CIT without the CI types which inherit from it.	<ol style="list-style-type: none"> 1. Name: name Description: Name of the source CIT. Is required?: Required Type: String 2. Name: mode Description: The type of update required for the current CI type. Is required?: Required Type: One of the following strings: <ol style="list-style-type: none"> a. insert: Use this only if the CI does not already exist. b. update: Use this only if the CI is known to exist. c. update_else_insert: If the CI exists, update it; otherwise, create a new CI. d. ignore: Do nothing with this CI type.
target_ci_type (integration > targetcis > source_ci_type -OR- integration > targetcis > source_ci_type_tree)	Defines a target CIT.	<ol style="list-style-type: none"> 1. Name: name Description: Target CI type name. Is required?: Required Type: String 2. Name: schema Description: The name of the schema that will be used to store this CI type at the target. Is required?: Not Required Type: String 3. Name: namespace Description: Indicates the namespace of this CI type on the target. Is required?: Not Required Type: String

Element Name and Path	Description	Attributes
targetprimarykey (integration > targetcis > source_ci_type) -OR- (integration > targetcis > source_ci_type_tree) -OR- (integration > targetrelations > link) -OR- (integration > targetrelations > source_link_type_tree)	Identifies target CIT primary key attributes.	
pkey (integration > targetcis > source_ci_type > targetprimarykey -OR- integration > targetcis > source_ci_type_tree > targetprimarykey -OR- (integration > targetrelations > link > targetprimarykey) -OR- integration > targetrelations > source_link_type_tree > targetprimarykey)	Identifies one primary key attribute. Required only if mode is update or insert_else_update .	

Element Name and Path	Description	Attributes
target_attribute (integration > targetcis > source_ci_type -OR- integration > targetcis > source_ci_type_tree -OR- integration > targetrelations > link -OR- integration > targetrelations > source_link_type_tree)	Defines the target CIT's attribute.	<ol style="list-style-type: none"> Name: name Description: Name of the target CIT's attribute. Is required?: Required Type: String Name: datatype Description: Data type of the target CIT's attribute. Is required?: Required Type: String Name: length Description: For string/char data types, integer size of target attribute. Is required?: Not Required Type: Integer Name: option Description: The conversion function to be applied to the value. Is required: False Type: One of the following strings: <ol style="list-style-type: none"> uppercase – Convert to uppercase lowercase – Convert to lowercase <p>If this attribute is empty, no conversion function will be applied.</p>

Element Name and Path	Description	Attributes
<p>map</p> <p>(integration > targetcis > source_ci_ type > target_attribute</p> <p>-OR-</p> <p>integration > targetcis > source_ci_type_tree > target_attribute)</p> <p>-OR-</p> <p>(integration > targetrelations > link > target_attribute</p> <p>-OR-</p> <p>integration > targetrelations > source_link_type_tree > target_attribute)</p>	<p>Specifies how to obtain the source CIT's attribute value.</p>	<ol style="list-style-type: none"> Name. type Description. The type of mapping between the source and target values. Is required. Required Type. One of the following strings: <ol style="list-style-type: none"> direct – Specifies a 1-to-1 mapping from source attribute's value to target attribute's value. compoundstring – Sub-elements are joined into a single string and the target attribute value is set. childattr – Sub-elements are one or more child CIT's attributes. Child CITs are defined as those with composition or containment relationship. constant – Static string Name. value Description. Constant string for type=constant Is required. Only required when type=constant Type. String Name. attr Description. Source attribute name for type=direct Is required. Only required when type=direct Type. String

Element Name and Path	Description	Attributes
<p>aggregation</p> <p>(integration > targetcis > source_ci_type > target_attribute > map</p> <p>-OR-</p> <p>integration > targetcis > source_ci_type_tree > target_attribute > map</p> <p>-OR-</p> <p>(integration > targetrelations > link > target_attribute > map</p> <p>-OR-</p> <p>integration > targetrelations > source_link_type_tree > target_attribute > map)</p> <p>Only valid when the map's type is childattr</p>	<p>Specifies how the source CI's child CI attribute values are combined into a single value to map to the target CI attribute. Optional.</p>	<p>Name: type</p> <p>Description. The type of aggregation function</p> <p>Is required?: Required</p> <p>Type. One of the following strings:</p> <ul style="list-style-type: none"> • csv – Concatenates all included values into a comma-separated list (numeric or string/character). • count – Returns a numeric count of all included values. • sum – Returns a numeric count of all included values. • average – Returns a numeric average of all included values. • min – Returns the lowest numeric/character included value. • max – Returns the highest numeric/character included value.

Element Name and Path	Description	Attributes
<p>source_child_ci_type (integration > targetcis > source_ci_type > target_attribute > map -OR- integration > targetcis > source_ci_type_tree > target_attribute > map -OR- (integration > targetrelations > link > target_attribute > map -OR- integration > targetrelations > source_link_type_tree > target_attribute > map)</p> <p>Only valid when the map's type is childattr.</p>	<p>Specifies from which connected CI the child attribute is taken.</p>	<ol style="list-style-type: none"> 1. Name. name Description. The type of the child CI Is required. Required Type. String 2. Name. source_attribute Description. The attribute of the child CI that is mapped. Is required. Required only if the childAttr aggregation type (which is on the same path) is not =count. Type. String

Element Name and Path	Description	Attributes
validation (integration > targetcis > source_ci_type > target_attribute > map -OR- integration > targetcis > source_ci_type_tree > target_attribute > map -OR- (integration > targetrelations > link > target_attribute > map -OR- integration > targetrelations > source_link_type_tree > target_attribute > map) Only valid when the map's type is childatt	Allows exclusion filtering of the source CI's child CIs based on attribute values. Used with the aggregation sub-element to achieve granularity of exactly which children attributes are mapped to the target CIT's attribute value. Optional.	<ol style="list-style-type: none"> 1. Name. minlength Description. Excludes strings shorter than the given value. Is Required?: Not required Type. Integer 2. Name. maxlength Description. Excludes strings longer than the given value. Is Required?: Not required Type. Integer 3. Name. minvalue Description. Excludes numbers smaller than the specified value. Is Required?: Not required Type. Numeric 4. Name. maxvalue Description. Excludes numbers greater than the specified value. Is Required?: Not required Type. Numeric
targetrelations (integration)	Container element for all relationship mappings. Optional.	

Element Name and Path	Description	Attributes
source_link_type_tree (integration > targetrelations)	Maps a source Relationship type without the types which inherit from it to a target Relationship. Mandatory only if targetrelation is present.	<ol style="list-style-type: none"> Name: name Description: Source relationship name. Is required?: Required Type: String Name: target_link_type Description: Target relationship name Is required?: Required Type: String Name: nameSpace Description: The namespace for the link that will be created on the target. Is required?: Not required Type: String Name: mode Description: The type of update required for the current link. Is required?: Required Type: One of the following strings: <ul style="list-style-type: none"> insert – Use this only if the CI does not already exist. update – Use this only if the CI is known to exist. update_else_insert – If the CI exists, update it; otherwise, create a new CI. ignore – Do nothing with this CI type. Name: source_ci_type_end1 Description: Source relationship's End1 CI type. Is required?: Required Type: String Name: source_ci_type_end2 Description: Source relationship's End2 CI type. Is required?: Required Type: String

Element Name and Path	Description	Attributes
link (integration > targetrelations)	Maps a source Relationship to a target Relationship. Mandatory only if targetrelation is present.	<ol style="list-style-type: none"> 1. Name: source_link_type Description: Source relationship name. Is Required?: Required Type: String 2. Name: target_link_type Description: Target relationship name. Is required?: Required Type: String 3. Name: nameSpace Description: The namespace for the link that will be created on the target. Is required?: Not required Type: String 4. Name: mode Description: The type of update required for the current link. Is required?: Required Type: On the following strings: <ul style="list-style-type: none"> ■ insert – Use this only if the CI does not already exist. ■ update – Use this only if the CI is known to exist. ■ update_else_insert – If the CI exists, update it; otherwise, create a new CI. ■ ignore – Do nothing with this CI type. 5. Name: source_ci_type_end1 Description: Source relationship's End1 CI type Is required?: Required Type: String 6. Name: source_ci_type_end2 Description: Source relationship's End2 CI type Is required?: Required Type: String

Element Name and Path	Description	Attributes
target_ci_type_end1 (integration > targetrelations > link -OR- integration > targetrelations > source_link_type_tree)	Target relationship's End1 CI type.	<ol style="list-style-type: none"> 1. Name: name Description: Name of the target relationship's End1 CI type. Is required?: Required Type: String 2. Name: superclass Description: Name of the End1 CI type's super-class. Is required?: Not required Type: String
target_ci_type_end2 (integration > targetrelations > link -OR- integration > targetrelations > source_link_type_tree)	Target relationship's End2 CI type.	<ol style="list-style-type: none"> 1. Name: name Description: Name of the target relationship's End2 CI type. Is required?: Required Type: String 2. Name: superclass Description: Name of the End2 CI type's super-class. Is required?: Not required Type: String

Mapping Results Schema

Element Name and Path	Description	Attributes
root	The root of the result document.	
data (root)	The root of the data itself.	
objects (root > data)	The root element for the objects to update.	
Object (root > data > objects)	Describes the update operation for a single object and all of its attributes.	<ol style="list-style-type: none"> Name: name Description: Name of the CI type Is required?: Required Type: String Name: mode Description: The type of update required for the current CI type. Is required?: Required Type: One of the following strings: <ol style="list-style-type: none"> insert – Use this only if the CI does not already exist. update – Use this only if the CI is known to exist. update_else_insert – If the CI exists, update it; otherwise, create a new CI. ignore – Do nothing with this CI type. Name: operation Description: The operation to perform with this CI. Is required: Required Type: One of the following strings: <ol style="list-style-type: none"> add – The CI should be added update – The CI should be updated delete – The CI should be deleted If no value is set, then the default value of add is used. Name: mamId Description: The ID of the object on the source RTSM. Is required?: Required Type: String

Element Name and Path	Description	Attributes
field (root > data > objects > Object -OR- root > data > links > link)	Describes the value of a single field for an object. The field's text is the new value in the field, and if the field contains a link, the value is the ID of one of the ends. Each end ID appears as an object (under <objects>).	<ol style="list-style-type: none"> Name: name Description: Name of the field. Is required?: Required Type: String Name: key Description: Specifies whether this field is a key for the object. Is required?: Required Type: Boolean Name: datatype Description: The type of the field. Is required?: Required Type: String Name: length Description: For string/character data types, this is the integer size of the target attribute. Is required?: Not Required Type: Integer

Element Name and Path	Description	Attributes
links (root > data)	The root element for the links to update.	<ol style="list-style-type: none"> Name: targetRelationshipClass Description: The name of the relationship (link) in the target system. Is required?: Required Type: String Name: targetParent Description: The type of first end of the link (parent). Is required?: Required Type: String Name: targetChild Description: The type of the second end of the link (child). Is required?: Required Type: String Name: mode Description: The type of update required for the current CI type. Is required?: Required Type: One of the following strings: <ol style="list-style-type: none"> insert – Use this only if the CI does not already exist. update – Use this only if the CI is known to exist. update_else_insert – If the CI exists, update it; otherwise, create a new CI. ignore – Do nothing with this CI type. Name: operation Description: The operation to perform with this CI. Is required?: Required Type: One of the following strings: <ol style="list-style-type: none"> add – The CI should be added update – The CI should be updated delete – The CI should be deleted If no value is set, then the default value of add is used. Name: mamId Description: The ID of the object on the source RTSM. Is required?: Required Type: String

Chapter 8

Viewing KPIs in External Applications

This chapter includes:

Set Up an Adapter to View KPIs in an External Application241

Set Up an Adapter to View KPIs in an External Application

This chapter explains how to retrieve the status and value of BSM KPIs when working in an external application. You can view the results in your application, or you can use the results in calculations, statistics, and so on.

This task describes how to set up the **BACKPIsAdapter**.

This task includes the following steps:

- "Create an Adapter Instance" below
- "View the Attributes in the CIT Manager" below
- "Define a TQL" on the next page
- "Consume KPIs in External Application" on the next page

1. Create an Adapter Instance

In BSM, in the integration point, create an adapter instance named **BACKPIsAdapter**. Use the following definitions:

- **Adapter.** Enter **BACKPIsAdapter**.
- **Is Integration Activated.** This checkbox should be selected.
- **Host name and IP.** Enter details of the BSM Gateway Server.
- **Port.** Enter 80.
- **Customer ID.** Enter 1.
- **Is CMS External.** Select **true** if you are working with an external CMS; select **false** if you are working with the RTSM.

In the second stage, choose the **Kpi** and **KpiObjective** CITs.

For details, see "Integration Point Pane " in the Data Flow Management Guide.

2. View the Attributes in the CIT Manager

The following attributes are used by the **Kpi** CIT:

- **KpiName.** The name of the KPI, for example, `Availability`, `Performance`.
- **Origin.** The origin of the KPIs, for example, `BSM`.

The following attributes are used by the **KpiObjective** CIT:

- **Origin.** The origin of the KPIs, for example, `BSM`.
- **KpiValue.** The numeric value of the KPI as calculated by the KPI business logic rule, for example, `0`, `100`, `3.2`. (This value is optional.)
- **KpiStatus.** The status of the KPI as calculated by the business logic rule (can be one of the following values: `ok`, `warning`, `minor`, `major`, `critical`, `none`).
- **KpiUnitOfMeasure.** The measurement unit of the `NumericValue` field, for example, `%`, `$`.

(This value is optional.)

- **LastModifiedTime.** The date when the status last changed.
- **KpiComparisonOperator.** This attribute contains enum operators: na, =, >, <, <=, >=.
- **KpiThresholdOk.** The range of values that defines the OK KPI status.
- **KpiThresholdWarning.** The range of values that defines the WARNING KPI status.
- **KpiThresholdMinor.** The range of values that defines the MINOR KPI status.
- **KpiThresholdMajor.** The range of values that defines MAJOR KPI status.
- **KpiThresholdCritical.** The range of values that defines the CRITICAL KPI status.

For details on working with the CIT Manager, see CI Type Manager and CI Type Attributes in the Modeling Guide.

For details on KPI rules, see "Business Rule Repository" in *Using Service Health*.

3. Define a TQL

Define a TQL using the **Kpi** and **KpiObjective** CITs. For details, see "Define a TQL Query" in the HP Universal CMDB Modeling Guide.

4. Consume KPIs in External Application

Using the API of your external application, send a query to run the relevant TQL in BSM, to retrieve the status and value of the federated KPIs.

Part 2

Using APIs

Chapter 9

Introduction to APIs

This chapter includes:

APIs Overview 245

APIs Overview

The following APIs are included with BSM:

- **UCMDB Web Service API.** Enables writing configuration item definitions and topological relations to the BSMRun-time Service Model (RTSM), and querying the information with TQL and ad hoc queries. For details, see ["RTSM \(HP Universal CMDB\) Web Service API" on page 258](#).
- **UCMDB Java API.** Explains how third-party or custom tools can use the Java API to extract data and calculations and to write data to the BSMRun-time Service Model (RTSM). For details, see ["HP Universal CMDB API" on page 246](#).
- **Data Flow Management Web Service API.** Enables managing probes, jobs, triggers and credentials for Data Flow Management. For details, see ["Data Flow Management API" on page 327](#).

Note: To gain the full value of the API documentation, it is recommended to access the online documentation. The PDF version does not have the links into the API documentation that is generated in html format.

Chapter 10

HP Universal CMDB API

This chapter includes:

Conventions	247
Using the HP Universal CMDB API	248
General Structure of an Application	249
Put the API Jar File in the Classpath	251
Create an Integration User	252
HP Universal CMDB API Reference	254
Use Cases	255
Examples	257

Conventions

Note: The HP Universal CMDB Web Service API can be used with the BSM Run-time Service Model (RTSM). References in this section to HP Universal CMDB (UCMDB) refer to the RTSM in the BSM context.

This chapter uses the following conventions:

- **UCMDB** refers to the Universal Configuration Management database itself. In the BSM context, it refers to the Run-time Service Model (RTSM). HP Universal CMDB refers to the application.
- UCMDB elements and method arguments are spelled in the case in which they are specified in the interfaces.

For full documentation on the available APIs, refer to the [HP UCMDB API Reference](#).

These files are located in the following folder:

```
\\<BSM Gateway Server root directory>\AppServer\  
webapps\site.war\amdocs\eng\APIs\UCMDB_JavaAPI\index.html
```

Using the HP Universal CMDB API

Note: Use this chapter in conjunction with the API Javadoc, available in the online Documentation Library.

The HP Universal CMDB API is used to integrate applications with the Run-time Service Model (RTSM). The API provides methods to:

- add, remove, and update CIs and relations in the RTSM
- retrieve information about the class model
- retrieve information from the UCMDB history
- run what-if scenarios
- retrieve information about configuration items and relationships

Methods for retrieving information about configuration items and relationships generally use the Topology Query Language (TQL). For details, see [Topology Query Language](#) in the Modeling Guide.

Users of the HP Universal CMDB API should be familiar with:

- The Java programming language
- HP Universal CMDB

This section includes the following topics:

- ["Uses of the API" below](#)
- ["Permissions" below](#)

Uses of the API

The API is used to fulfill a number of business requirements. For example, a third-party system can query the class model for information about available configuration items (CIs). For more use cases, see ["Use Cases" on page 255](#).

Permissions

The administrator provides login credentials for connecting with the API. The API client needs the user name and password of an integration user defined in the RTSM. These users do not represent human users of the RTSM, but rather applications that connect to the RTSM.

Caution: The API client can also work with regular users as long as they have API authentication permission. However, this option is not recommended.

For details, see ["Create an Integration User" on page 252](#).

General Structure of an Application

There is only one static factory, the `UcmdbServiceFactory`. This factory is the entry point for an application. The `UcmdbServiceFactory` exposes `getServiceProvider` methods. These methods return an instance of the **UcmdbServiceProvider** interface.

The client creates other objects using interface methods. For example, to create a new query definition, the client:

1. Gets the query service from the main RTSM service object
2. Gets a query factory object from the service object
3. Gets a new query definition from the factory

```
UcmdbServiceProvider provider =
    UcmdbServiceFactory.getServiceProvider(HOST_NAME, PORT);
UcmdbService ucmdbService =
    provider.connect(provider.createCredentials(USERNAME,
        PASSWORD), provider.createClientContext("Test"));
TopologyQueryService queryService =
    ucmdbService.getTopologyQueryService();
TopologyQueryFactory factory = queryService.getFactory();
QueryDefinition queryDefinition = factory.createQueryDefinition
    ("Test Query");
queryDefinition.addNode("Node").ofType("host");
Topology topology = queryService.executeQuery(queryDefinition);
System.out.println("There are " + topology.getAllCIs().size() +
    " hosts in uCMDB");
```

The services available from **UcmdbService** are:

Service Methods	Use
<code>getClassModelService</code>	Information about types of CIs and relations
<code>getConfigurationService</code>	Infrastructure settings management, for server configuration
<code>getDDMConfigurationService</code>	Configure the Data Flow Management system
<code>getDDMManagementService</code>	Analyze and view the progress, results, and errors of the Data Flow Management system
<code>getHistoryService</code>	Information about history of monitored CIs (changes, removals, and so on)
<code>getImpactAnalysisService</code>	Run impact analysis scenario (also known as correlation).

Service Methods	Use
getQueryManagementService	Manage access to queries - save, delete, list existing. Also provides query validation and queries dependencies discovery.
getResourceBundleManagementService	Resource tagging ("bundling" services. Allows explicit creation of new tags and removal of tags from all tagged resources.
getStateService	Provide services for managing states (list, add, remove, and so on)
getSoftwareSignatureService	Define software items to be discovered by the Discovery and Dependency Management system
getSnapshotService	Provide services for managing snapshots (get, save, compare, and so on)
getTopologyQueryService	Get information about the IT universe
getTopologyUpdateService	Change information in the IT universe
getViewService	View execution service (execute definition, execute saved) and management service (save, delete, list existing). Also provides view validation and dependencies discovery.
getViewArchiveService	View result archiving services. Allows saving the current view result and retrieving previously saved results.
SystemHealthService	Provide system health services (basic system performance indicators, capacity and availability metrics)

The client communicates with the server over HTTP.

Put the API Jar File in the Classpath

The use of this API set requires the file **ucmdb-api.jar**. Get the jar from the **<HP BSM root directory>\lib** folder.

Put the jar file in the classpath before compiling or running your application.

Note: Usage of the UCMDB Java API Jar requires you to have JRE version 6 or later installed.

Create an Integration User

You can create a dedicated user for integrations between other products and BSM. This user enables a product that uses the BSM client SDK to be authenticated in the server SDK and execute the APIs. Applications written with this API set must log on with integration user credentials.

Caution: It is also possible to connect with a regular UCMDB user, (for instance, admin); however, this option is not recommended. To connect with a UCMDB user, you must grant it API authentication permission.

To create an integration user:

1. Launch the Web browser and enter the server address, as follows.
`http://localhost.<domain_name>:21212/jmx-console.`
You may have to log in with a user name and password.
2. Under UCMDB, click **service=UCMDB Security Services**.
3. Locate the **CreateIntegrationUser** operation. This method accepts the following parameters:

- **customerId**. The customer ID.
- **username**. The integration user's name.
- **password**. The integration user's password.
- **dataStoreOrigin**. The name of the product that is going to use this integration user.

The following operations are useful for integration user management:

- **DeleteIntegrationUser**. Deletes the given integration user.
- **ExportIntegrationUser**. Exports the integration user to an XML file in the given path (on the server machine).
- **getIntegrationUser**. Displays the integration user information.
- **changeIntegrationUserPassword**. Changes the integration user's password.
- **canUserAuthenticate**. **isIntegrationUser** is **true**: can the integration user authenticate with the given credentials?

4. Click **Invoke**.
Either create more users, or close the JMX console.
5. Log on to BSM as an administrator.
6. From the **Administration** tab, run **Package Manager**.
7. Click the **New** icon.
8. Enter a name for the new package, and click **Next**.
9. In the Resource Selection tab, under **Administration**, click **Integration Users**.
10. Select a user or users that you created using the JMX console.

11. Click **Next** and then **Finish**. Your new package appears in the Package Name list in Package Manager.
12. Deploy the package to the users who will run the API applications.

For details, see Deploy a Package in the *RTSM Administration Guide*.

Note:

The integration user is per customer. To create a stronger integration user for cross-customer usage, use a **systemUser** with the **isSuperIntegrationUser** flag set to **true**. Use the **systemUser** methods (**createSystemUser**, **removeSystemUser**, **showAllSystemUsers**, **changeSystemUserPassword**, **canSuperIntegrationUserAuthenticate**, and so on).

There are two out-of-the-box system users; it is recommended to change their passwords after installation using the **changeSystemUserPassword** method.

- **sysadmin/sysadmin**
- **UISysadmin/UISysadmin** (This user is also the Super Integration User **SuperIntegrationUser**).

If you change the UISysadmin password using **changeSystemUserPassword**, you must execute the following method: in the JMX Console, locate the **UCMDB-UI:name=UCMDB Integration** service. Run **setCMDBSuperIntegrationUser** with the user name and new password of the integration user.

HP Universal CMDB API Reference

These files are located in the following folder:

**\\<BSM Gateway Server root directory>\AppServer\
webapps\site.war\amdocs\eng\APIs\UCMDB_JavaAPI \index.html**

Use Cases

The following use cases assume two systems:

- HP Universal CMDB server
- A third-party system that contains a repository of configuration items

This section includes the following topics:

- ["Populating the RTSM" below](#)
- ["Querying the RTSM" below](#)
- ["Querying the Class Model" below](#)
- ["Analyzing Change Impact " below](#)

Populating the RTSM

Use cases:

- A third-party asset management updates the RTSM with information available only in asset management
- A number of third-party systems populate the RTSM to create a central CMDB that can track changes and perform impact analysis
- A third-party system creates Configuration Items and Relations according to third-party business logic, to leverage the RTSM query capabilities

Querying the RTSM

Use cases:

- A third-party system gets the Configuration Items and Relations that represent the SAP system by retrieving the results of the SAP TQL
- A third-party system gets the list of Oracle servers that have been added or changed in the last five hours
- A third-party system gets the list of servers whose host name contains the `lab` substring
- A third-party system finds the elements related to a given CI by getting its neighbors

Querying the Class Model

Use cases:

- A third-party system enables users to specify the set of data to be retrieved from the RTSM. A user interface can be built over the class model to show users the possible properties and prompt them for required data. The user can then choose the information to be retrieved.
- A third-party system explores the class model when the user cannot access the BSM user interface.

Analyzing Change Impact

Use case:

A third-party system outputs a list of the business services that could be impacted by a change on a specified host.

Examples

See the following code samples:

- Create a Connection
- Create and Execute an Ad-Hoc Query
- Create and Execute a View
- Add and Delete Data
- Execute an Impact Analysis
- Query the Class Model
- Query a History Sample

These files are located in the following directory:

**\\<BSM Gateway Server root directory>\AppServer\
webapps\site.war\amdocs\eng\APIs\JavaSDK_Samples**

Chapter 11

RTSM (HP Universal CMDB) Web Service API

This chapter includes:

Conventions	259
RTSM (HP Universal CMDB) Web Service API Overview	260
RTSM (HP Universal CMDB) Web Service API Reference	262
Call the Web Service	263
Query the RTSM	264
Update the RTSM	267
Query the BSM Class Model	269
Query for Impact Analysis	271
UCMDB General Parameters	272
UCMDB Output Parameters	275
UCMDB Query Methods	277
UCMDB Update Methods	288
UCMDB Impact Analysis Methods	291
Use Cases	293
Examples	295

Conventions

Note: The HP Universal CMDB Web Service API can be used with the BSMRun-time Service Model (RTSM). References in this section to HP Universal CMDB (UCMDB) refer to the RTSM in the BSM context.

This chapter uses the following conventions:

- **UCMDB** refers to the Universal Configuration Management database itself. In the BSM context, it refers to the Run-time Service Model (RTSM). HP Universal CMDB refers to the application.
- BSM elements and method arguments are spelled in the case in which they are specified in the schema. An element or argument to a method is not capitalized. For example, a `relation` is an element of type `Relation` passed to a method.

For full documentation on the request and response structures, refer to the [HP UCMDB Web Service API Reference](#). These files are located in the following folder:

```
\\<BSM root directory>\AppServer\  
webapps\site.war\amdocs\eng\APIs\CMDB_Schema  
\webframe.html
```

RTSM (HP Universal CMDB) Web Service API Overview

Note to HP Software-as-a-Service customers: For details on how to use the UCMDB Web Service API in an HP Software-as-a-Service environment, contact HP Software-as-a-Service Support.

Note: Use this chapter in conjunction with the UCMDB schema documentation, available in the online Documentation Library.

The HP Universal CMDB Web Service API is used to integrate applications with the BSMRun-time Service Model (RTSM). The API provides methods to:

- add, remove, and update CIs and relations in the RTSM
- retrieve information about the class model
- retrieve impact analyses
- retrieve information about configuration items and relationships
- manage credentials: view, add, update, and remove
- manage jobs: view status, activate, and deactivate
- manage Probe ranges: view, add, and update
- manage triggers: add or remove a trigger CI, and add, remove, or disable a trigger TQL
- view general data on domains and Probes

Methods for retrieving information about configuration items and relationships generally use the Topology Query Language (TQL). For details, see Topology Query Language in the Modeling Guide.

Users of the HP Universal CMDB Web Service API should be familiar with:

- The SOAP specification
- An object-oriented programming language such as C++, C# or Java
- HP Universal CMDB
- Data Flow Management

This section includes the following topics:

- ["Uses of the API" below](#)
- ["Permissions" on the next page](#)

Uses of the API

The API is used to fulfill a number of business requirements. For example:

- A third-party system can query the class model for information about available configuration items (CIs).

- A third-party asset management tool can update the RTSM with information available only to that tool, thereby unifying its data with data collected by HP applications.
- A number of third-party systems can populate the RTSM to create a central RTSM that can track changes and perform impact analysis.
- A third-party system can create entities and relations according to its business logic, and then write the data to the RTSM to take advantage of the RTSM query capabilities.
- Other systems, such as the Release Control (CCM) system, can use the Impact Analysis methods for change analysis.

Permissions

The administrator provides login credentials for connecting with the Web Service. The required credentials depend on whether you are using HP Universal CMDB as a standalone application or from within BSM:

- HP Universal CMDB **standalone**. Log in using the credentials of a BSM user who has been granted permissions on the discovery and integration resources.

For details, see "Security Manager Page" in the *HP Universal CMDB Administration Guide*.

- HP Universal CMDB **embedded in** BSM. Log in using the credentials of a BSM user. The user must have been granted the relevant permissions on the HP Universal CMDB resource in BSM.

When permissions are assigned through HP Universal CMDB, the permission levels are View, Update, and Execute. When they are assigned using BSM, the levels are View and Update, where Update also includes Execution. To view the permissions required for each operation, see each operation's request documentation.

RTSM (HP Universal CMDB) Web Service API Reference

For full documentation on the request and response structures, refer to the [HP UCMDB Web Service API Reference](#). These files are located in the following folder:

**\\<BSM root directory>\AppServer\
webapps\site.war\amdocs\eng\APIs\CMDB_Schema\webframe.html**

Call the Web Service

You use standard SOAP programming techniques in the HP Universal CMDB Web Service to enable calling server-side methods. If the statement cannot be parsed or if there is a problem invoking the method, the API methods throw a `SoapFault` exception. When a `SoapFault` exception is thrown, BSM populates one or more of the error message, error code, and exception message fields. If there is no error, the results of the invocation are returned.

SOAP programmers can access the WSDL at:

`http://<server>[:port]/axis2/services/UcmdbService?wsdl`

The port specification is only necessary for non-standard installations. Consult your system administrator for the correct port number.

The URL for calling the service is:

`http://<server>[:port]/axis2/services/UcmdbService`

For examples of connecting to the RTSM, see ["Use Cases" on page 293](#).

Query the RTSM

The RTSM is queried using the APIs described in ["UCMDB Query Methods" on page 277](#).

The queries and the returned RTSM elements always contain real BSM IDs.

For examples of the use of the query methods, see ["Query Example" on page 296](#).

This section includes the following topics:

- ["Just In Time Response Calculation" below](#)
- ["Processing Large Responses" below](#)
- ["Specifying Properties to Return" on the next page](#)
- ["Concrete Properties" on the next page](#)
- ["Derived Properties" on page 266](#)
- ["Naming Properties" on page 266](#)
- ["Other Property Specification Elements" on page 266](#)

Just In Time Response Calculation

For all query methods, the BSM server calculates the values requested by the query method when the request is received, and returns results based on the latest data. The result is always calculated at the time the request is received, even if the TQL query is active and there exists a previously calculated result. Therefore, the results of running a query returned to the client application may be different to the results of the same query displayed on the user interface.

Tip: If your application uses the results of a given query more than once and the data is not expected to change significantly between uses of the result data, you can improve performance by having the client application store the data rather than repeatedly running the query.

Processing Large Responses

The response to a query always includes the structures for the data requested by the query method, even if no actual data is being transmitted. For many methods where the data is a collection or map, the response also includes the `ChunkInfo` structure, comprised of `chunksKey` and `numberOfChunks`. The `numberOfChunks` field indicates the number of chunks containing data that must be retrieved.

The maximum transmission size of data is set by the system administrator. If the data returned from the query is larger than the maximum size, the data structures in the first response contain no meaningful information, and the value of the `numberOfChunks` field is 2 or greater. If the data is not larger than the maximum, the `numberOfChunks` field is 0 (zero), and the data is transmitted in the first response. Therefore, in processing a response, check the `numberOfChunks` value first. If it is greater than 1, discard the data in the transmission and request the chunks of data. Otherwise, use the data in the response.

For information on handling chunked data, see ["pullTopologyMapChunks" on page 285](#) and ["releaseChunks" on page 287](#).

Specifying Properties to Return

CIs and relations generally have many properties. Some methods that return collections or graphs of these items accept input parameters that specify which property values to return with each item that matches the query. The RTSM does not return empty properties. Therefore, the response to a query may have fewer properties than requested in the query.

This section describes the types of sets used to specify the properties to return.

Properties can be referenced in two ways:

- By their names
- By using names of predefined properties rules. Predefined properties rules are used by the RTSM to create a list of real property names.

When an application references properties by name, it passes a `PropertiesList` element.

Tip: Whenever possible, use `PropertiesList` to specify the names of the properties in which you are interested, rather than a rule-based set. The use of predefined properties rules nearly always results in returning more properties than needed, and bears a performance price.

There are two types of predefined properties: qualifier properties and simple properties.

- **Qualifier properties.** Use when the client application should pass a `QualifierProperties` element (a list of qualifiers that can be applied to properties). The RTSM converts the list of qualifiers passed by the client application to the list of the properties to which at least one of the qualifiers applies. The values of these properties are returned with the `CI` or `Relation` elements.
- **Simple properties.** To use simple rule-based properties, the client application passes a `SimplePredefinedProperty` or `SimpleTypedPredefinedProperty` element. These elements contain the name of the rule by which the RTSM generates the list of properties to return. The rules that can be specified in a `SimplePredefinedProperty` or `SimpleTypedPredefinedProperty` element are `CONCRETE`, `DERIVED`, and `NAMING`.

Concrete Properties

Concrete properties are the set of properties defined for the specified CIT. The properties added by derived classes are not returned for instances of those derived classes.

A collection of instances returned by a method may consist of instances of a CIT specified in the method invocation and instances of CITs that inherit from that CIT. The derived CITs inherit the properties of the specified CIT. In addition, the derived CITs extend the parent CIT by adding properties.

Example of Concrete Properties:

CIT `T1` has properties `P1` and `P2`. CIT `T11` inherits from `T1` and extends `T1` with properties `P21` and `P22`.

The collection of CIs of type `T1` includes the instances of `T1` and `T11`. The concrete properties of all instances in this collection are `P1` and `P2`.

Derived Properties

Derived properties are the set of properties defined for the specified CIT and, for each derived CIT, the properties added by the derived CIT.

Example of Derived Properties:

Continuing the example from concrete properties, the derived properties of instances of T1 are P1 and P2. The derived properties of instances of T11 are P1, P2, P21, and P22.

Naming Properties

The naming properties are `display_label` and `data_name`.

Other Property Specification Elements

- **PredefinedProperties**

`PredefinedProperties` can contain a `QualifierProperties` element and a `SimplePredefinedProperty` element for each of the other possible rules. A `PredefinedProperties` set does not necessarily contain all types of lists.

- **PredefinedTypedProperties**

`PredefinedTypedProperties` is used to apply a different set of properties to each CIT. `PredefinedTypedProperties` can contain a `QualifierProperties` element and a `SimpleTypedPredefinedProperty` element for each of the other applicable rules. Because `PredefinedTypedProperties` is applied to each CIT individually, derived properties are not relevant. A `PredefinedProperties` set does not necessarily contain all applicable types of lists.

- **CustomProperties**

`CustomProperties` can contain any combination of the basic `PropertiesList` and the rule-based property lists. The properties filter is the union of all the properties returned by all the lists.

- **CustomTypedProperties**

`CustomTypedProperties` can contain any combination of the basic `PropertiesList` and the applicable rule-based property lists. The properties filter is the union of all the properties returned by all the lists.

- **TypedProperties**

`TypedProperties` is used to pass a different set of properties for each CIT. `TypedProperties` is a collection of pairs composed of type names and properties sets of all types. Each properties set is applied only to the corresponding type.

Update the RTSM

You update the RTSM with the update APIs. For details of the API methods, see ["UCMDB Update Methods" on page 288](#). For examples of the use of the update methods, see ["Update Example" on page 308](#).

This task includes the following steps:

- ["Update the RTSM" above](#)
- ["Use of ID Types with Update Methods" below](#)

UCMDB Update Parameters

This topic describes the parameters used only by the service's update methods. For details, see the [schema documentation](#).

- **CIsAndRelationsUpdates**

The `CIsAndRelationsUpdates` type consists of `CIsForUpdate`, `relationsForUpdate`, `referencedRelations`, and `referencedCIs`. A `CIsAndRelationsUpdates` instance does not necessarily include all three elements.

`CIsForUpdate` is a `CIs` collection. `relationsForUpdate` is a `Relations` collection. The `CI` and `relation` elements in the collections have a `props` element. When creating a `CI` or `relation`, properties that have either the `required` attribute or the `key` attribute in the `CI` Type definition must be populated with values. The items in these collections are updated or created by the method.

`referencedCIs` and `referencedRelations` are collections of `CIs` that are already defined in the RTSM. The elements in the collection are identified with a temporary ID in conjunction with all the key properties. These items are used to resolve the identities of `CIs` and `relations` for update. They are never created or updated by the method.

Each of the `CI` and `relation` elements in these collections has a `properties` collection. New items are created with the property values in these collections.

Use of ID Types with Update Methods

The following describes ID `CITs`, and `CIs` and `relations`. When the ID is not a real RTSM ID, the type and key attributes are required.

- **Deleting or Updating Configuration Items**

A temporary or empty ID may be used by the client when calling a method to delete or update an item. In this case, the `CI` type and the ["Key Attributes"](#) that identify the `CI` must be set.

- **Deleting or Updating Relations**

When deleting or updating `relations`, the `relation` ID can be empty, temporary, or real.

If a `CI`'s ID is temporary, the `CI` must be passed in the `referencedCIs` collection and its key attributes must be specified. For details, see `referencedCIs` in ["CIsAndRelationsUpdates" above](#).

- **Inserting New Configuration Items into the RTSM**

It is possible to use either an empty ID or a temporary ID to insert a new CI. However, if the ID is empty, the server cannot return the real RTSM ID in the structure `createIDsMap` because there is no `clientID`. For details, see ["addCIsAndRelations" on page 288](#) and ["UCMDB Query Methods" on page 277](#).

- **Inserting New Relations into the RTSM**

The relation ID can be either temporary or empty. However, if the relation is new but the configuration items on either end of the relation are already defined in the RTSM, then those CIs that already exist must be identified by a real RTSM ID or be specified in a `referencedCIs` collection.

Query the BSM Class Model

The class model methods return information about CITs and relations. The class model is configured using the CI Type Manager. For details, see [CI Type Manager](#) in the Modeling Guide.

For examples of the use of the class model methods, see ["Class Model Example" on page 312](#).

This section provides information on the following methods that return information about CITs and relations:

- ["getClassAncestors" below](#)
- ["getAllClassesHierarchy" below](#)
- ["getCmdbClassDefinition" on the next page](#)

getClassAncestors

The `getClassAncestors` method retrieves the path between the given CIT and its root, including the root.

.

Input

Parameter	Comment
cmdbContext	For details, see "CmdbContext" on page 272 .
className	The type name. For details, see "Type Name" on page 273 .

Output

Parameter	Comment
classHierarchy	A collection of pairs of class names and parent class name.
comments	For internal use only.

getAllClassesHierarchy

The `getAllClassesHierarchy` method retrieves the entire class model tree.

.

Input

Parameter	Comment
cmdbContext	For details, see "CmdbContext" on page 272 .

Output

Parameter	Comment
classesHierarchy	A collection of pairs of class name and parent class name.
comments	For internal use only.

getCmdbClassDefinition

The `getCmdbClassDefinition` method retrieves information about the specified class.

If you use `getCmdbClassDefinition` to retrieve the key attributes, you must also query the parent classes up to the base class. `getCmdbClassDefinition` identifies as key attributes only those attributes with the `ID_ATTRIBUTE` set in the class definition specified by `className`. Inherited key attributes are not recognized as key attributes of the specified class. Therefore, the complete list of key attributes for the specified class is the union of all the keys of the class and of all its parents, up to the root.

Input

Parameter	Comment
cmdbContext	For details, see "CmdbContext" on page 272 .
className	The type name. For details, see "UCMDB General Parameters" on page 272 .

Output

Parameter	Comment
cmdbClass	The class definition, consisting of <code>name</code> , <code>classType</code> , <code>displayLabel</code> , <code>description</code> , <code>parentName</code> , <code>qualifiers</code> , and <code>attributes</code> .
comments	For internal use only.

Query for Impact Analysis

The `Identifier` in the impact analysis methods points to the service's response data. It is unique for the current response and is discarded from the server's memory cache after 10 minutes of non-use.

For examples of the use of the impact analysis methods, see ["Impact Analysis Example" on page 314](#).

UCMDB General Parameters

This section describes the most common parameters of the service's methods. For details, refer to the [schema documentation](#).

This section includes the following topics:

- ["CmdbContext" below](#)
- ["ID" below](#)
- ["Key Attributes" below](#)
- ["ID Types" below](#)
- ["CIProperties" on the next page](#)
- ["Type Name" on the next page](#)
- ["Configuration Item \(CI\)" on the next page](#)
- ["Relation" on the next page](#)

CmdbContext

All UCMDB Web Service API service invocations require a `CmdbContext` argument.

`CmdbContext` is a `callerApplication` string that identifies the application that invokes the service. `CmdbContext` is used for logging and troubleshooting.

ID

Every `CI` and `Relation` has an `ID` field. It consists of a case-sensitive ID string and an optional `temp` flag, indicating whether the ID is temporary.

Key Attributes

For identifying a `CI` or `Relation` in some contexts, key attributes can be used in place of a RTSM ID. Key attributes are those attributes with the `ID_ATTRIBUTE` set in the class definition.

In the user interface, the key attributes have a key icon next to them in the list of Configuration Item Type attributes in the user interface. For details, see [Add/Edit Attribute Dialog Box](#) in the Modeling Guide. For information about identifying the key attributes from within the API client application, see ["getCmdbClassDefinition" on page 270](#).

ID Types

An `ID` element can contain a real ID, or a temporary ID.

A real ID is a string assigned by the RTSM that identifies an entity in the database. A temporary ID can be any string that is unique in the current request.

A temporary ID can be assigned by the client and often represents the ID of the CI as stored by the client. It does not necessarily represent an entity already created in the RTSM. When a temporary ID is passed by the client, if the RTSM can identify an existing data configuration item using the CI key properties, that CI is used as appropriate for the context as though it had been identified with a real ID.

CIProperties

A `CIProperties` element is composed of collections, each containing a sequence of name-value elements that specify properties of the type indicated by the collection name. None of the collections are required, so the `CIProperties` element can contain any combination of collections.

`CIProperties` are used by `CI` and `Relation` elements. For details, see ["Configuration Item \(CI\)" below](#) and ["Relation" below](#).

The properties collections are:

- `dateProps` - collection of `DateProp` elements
- `doubleProps` - collection of `DoubleProp` elements
- `floatProps` - collection of `FloatProp` elements
- `intListProps` - collection of `intListProp` elements
- `intProps` - collection of `IntProp` elements
- `strProps` - collection of `StrProp` elements
- `strListProps` - collection of `StrListProp` elements
- `longProps` - collection of `LongProp` elements
- `bytesProps` - collection of `BytesProp` elements
- `xmlProps` - collection of `XmlProp` elements

Type Name

The type name is the class name of a configuration item type or relation type. The type name is used in code to refer to the class. It should not be confused with the display name, which is seen on the user interface where the class is mentioned, but which is meaningless in code.

Configuration Item (CI)

A `CI` element is composed of an `ID`, a `type`, and a `props` collection.

When using ["UCMDB Update Methods"](#) to update a `CI`, the `ID` element can contain a real RTSM ID or a client-assigned temporary ID. If a temporary ID is used, set the `temp` flag to true. When deleting an item, the `ID` can be empty. ["UCMDB Query Methods"](#) take real IDs as input parameters and return real IDs in the query results.

The `type` can be any type name defined in the CI Type Manager. For details, see CI Type Manager in the Modeling Guide.

The `props` element is a `CIProperties` collection. For details, see ["UCMDB General Parameters" on the previous page](#).

Relation

A `Relation` is an entity that links two configuration items. A `Relation` element is composed of an `ID`, a `type`, the identifiers of the two items being linked (`end1ID` and `end2ID`), and a `props` collection.

When using ["UCMDB Update Methods"](#) to update a `Relation`, the value of the `Relation`'s `ID` can be a real RTSM ID or a temporary ID. When deleting an item, the `ID` can be empty. ["UCMDB Query Methods"](#) take real `IDs` as input parameters and return real `IDs` in the query results.

The relation type is the `Type Name` of the BSM class from which the relation is instantiated. The type can be any of the relation types defined in the RTSM. For further information on classes or types, see ["Query the BSM Class Model" on page 269](#).

For details, see CI Type Manager in the Modeling Guide.

The two relation end IDs must not be empty IDs because they are used to create the ID of the current relation. However, they both can have temporary IDs assigned to them by the client.

The `props` element is a `CIProperties` collection. For details, see ["CIProperties" on the previous page](#).

UCMDB Output Parameters

This section describes the most common output parameters of the service methods. For details, refer to the [schema documentation](#).

This section includes the following topics:

- ["CIs" below](#)
- ["ShallowRelation" below](#)
- ["Topology" below](#)
- ["CINode" below](#)
- ["RelationNode" below](#)
- ["TopologyMap" below](#)
- ["ChunkInfo" on the next page](#)

CIs

`CIs` is a collection of CI elements.

ShallowRelation

A `ShallowRelation` is an entity that links two configuration items, composed of an `ID`, a `type`, and the identifiers of the two items being linked (`end1ID` and `end2ID`). The relation type is the `Type Name` of the RTSM class from which the relation is instantiated. The type can be any of the relation types defined in the RTSM.

Topology

`Topology` is a graph of CI elements and relations. A `Topology` consists of a `CIs` collection and a `Relations` collection containing one or more `Relation` elements.

CINode

`CINode` is composed of a `CIs` collection with a `label`. The `label` in the `CINode` is the label defined in the node of the TQL used in the query.

RelationNode

`RelationNode` is a set of `Relations` collections with a `label`. The `label` in the `RelationNode` is the label defined in the node of the TQL used in the query.

TopologyMap

`TopologyMap` is the output of a query calculation that matches a TQL query. The `labels` in the `TopologyMap` are the node labels defined in the TQL used in the query.

The data of `TopologyMap` is returned in the following form:

- `CINodes`. This is one or more `CINode` (see ["CINode" above](#)).
- `relationNodes`. This is one or more `RelationNode` (see ["RelationNode" above](#)).

The `labels` in these two structures order the lists of configuration items and relations.

ChunkInfo

When a query returns a large amount of data, the server stores the data, divided into segments called chunks. The information the client uses to retrieve the chunked data is located in the `ChunkInfo` structure returned by the query. `ChunkInfo` is composed of the `numberOfChunks` that must be retrieved and the `chunksKey`. The `chunksKey` is a unique identifier of the data on the server for this specific query invocation.

For more information, see ["Processing Large Responses" on page 264](#).

UCMDB Query Methods

This section provides information on the following methods:

- ["executeTopologyQueryByNameWithParameters" below](#)
- ["executeTopologyQueryWithParameters" on the next page](#)
- ["getChangedCIs" on page 279](#)
- ["getCINeighbours" on page 279](#)
- ["getCIsById" on page 280](#)
- ["getCIsByType" on page 280](#)
- ["getFilteredCIsByType" on page 281](#)
- ["getQueryNameOfView" on page 284](#)
- ["getTopologyQueryExistingResultByName" on page 284](#)
- ["getTopologyQueryResultCountByName" on page 285](#)
- ["pullTopologyMapChunks" on page 285](#)
- ["releaseChunks" on page 287](#)

executeTopologyQueryByNameWithParameters

The `executeTopologyQueryByNameWithParameters` method retrieves a `topologyMap` element that matches the specified parameterized query.

The values for the query parameters are passed in the `parameterizedNodes` argument. The specified TQL must have unique labels defined for each `CINode` and each `relationNode` or the method invocation fails.

Input

Parameter	Comment
<code>cmdbContext</code>	For details, see "CmdbContext" on page 272 .
<code>queryName</code>	The name of the parameterized TQL in the RTSM for which to get the map.
<code>parameterizedNodes</code>	The conditions each node must meet to be included in the query results.
<code>queryTypedProperties</code>	A collection of sets of properties to retrieve to items of a specific Configuration Item Type.

Output

Parameter	Comment
topologyMap	For details, see "TopologyMap" on page 275 .
chunkInfo	For details, see: "ChunkInfo" on page 276 and "Processing Large Responses" on page 264 .

executeTopologyQueryWithParameters

The `executeTopologyQueryWithParameters` method retrieves a `topologyMap` element that matches the parameterized query.

The query is passed in the `queryXML` argument. The values for the query parameters are passed in the `parameterizedNodes` argument. The TQL must have unique labels defined for each `CINode` and each `relationNode`.

The `executeTopologyQueryWithParameters` method is used to pass ad-hoc queries, rather than accessing a query defined in the RTSM. You can use this method when you do not have access to the BSM user interface to define a query, or when you do not want to save the query to the database.

To use an exported TQL as the input to this method, do the following:

1. Launch the Web browser and enter the following address:
`http://localhost:8080/jmx-console`.

You may have to log in with a user name and password. The default is **`sysadmin/sysadmin`**

2. Click **UCMDB:service=TQL Services**.
3. Locate the **exportTql** operation.
 - In the **customerId** parameter box, enter **1** (the default).
 - In the **patternName** parameter box, enter a valid TQL name.
4. Click **Invoke**.

Input

Parameter	Comment
cmdbContext	For details, see "CmdbContext" on page 272 .
queryXML	An XML string representing a TQL without resource tags.
parameterizedNodes	The conditions each node must meet to be included in the query results.

Output

Parameter	Comment
topologyMap	For details, see "TopologyMap" on page 275 .
chunkInfo	For details, see "ChunkInfo" on page 276 and "Processing Large Responses" on page 264 .

getChangedCIs

The `getChangedCIs` method returns the change data for all CIs related to the specified CIs.

Input

Parameter	Comment
cmdbContext	For details, see "CmdbContext" on page 272 .
ids	The list of the IDs of the root CIs whose related CIs are checked for changes. Only real RTSM IDs are valid in this collection.
fromDate	The beginning of the period in which to check if CIs changed.
toDate	The end of the period in which to check if CIs changed.

Output

Parameter	Comment
changeDataInfo	Zero or more collections of <code>ChangedDataInfo</code> elements.

getCINeighbours

The `getCINeighbours` method returns the immediate neighbors of the specified CI.

For example, if the query is on the neighbors of CI A, and CI A contains CI B which uses CI C, CI B is returned, but CI C is not. That is, only neighbors of the specified type are returned.

.

Input

Parameter	Comment
cmdbContext	For details, see "CmdbContext" on page 272 .
ID	The ID of the CI with which to retrieve the neighbors. This must be a real RTSM ID.

Parameter	Comment
neighbourType	The CIT name of the neighbors to retrieve. Neighbors of the specified type and of types derived from that type are returned. For details, see "Type Name" on page 273 .
CIProperties	The data to be returned on each configuration item, called the Query Layout in the user interface. For details, see "TypedProperties" on page 266 .
relationProperties	The data to be returned on each relation (called the Query Layout in the user interface). For details, see "TypedProperties" on page 266 .

Output

Parameter	Comment
topology	For details, see "Topology" on page 275 .
comments	For internal use only.

getCIsByID

The `getCIsByID` method retrieves configuration items by their RTSM IDs.

Input

Parameter	Comment
cmdbContext	For details, see "CmdbContext" on page 272 .
CIsTypedProperties	A typed properties collection. For details, see "Other Property Specification Elements" on page 266 .
IDs	Only real RTSM IDs are valid in this collection.

Output

Parameter	Comment
CIs	Collection of CI elements.
chunkInfo	For details, see: "ChunkInfo" on page 276 and "Processing Large Responses" on page 264 .

getCIsByType

The `getCIsByType` method returns the collection of configuration items of the specified type and of all types that inherit from the specified type.

Input

Parameter	Comment
cmdbContext	For details, see "CmdbContext" on page 272 .
type	The class name. For details, see "Type Name" on page 273 .
properties	The data to be returned on each configuration item. For details, see "CustomProperties" on page 266 .

Output

Parameter	Comment
CIs	Collection of CI elements.
chunkInfo	For details, see: "ChunkInfo" on page 276 and "Processing Large Responses" on page 264 .

getFilteredCIsByType

The `getFilteredCIsByType` method retrieves the CIs of the specified type that meet the conditions used by the method. A condition is comprised of:

- a name field containing the name of a property
- an operator field containing a comparison operator
- an optional value field containing a value or list of values

Together, they form a Boolean expression:

```
<item>.property.value [operator] <condition>.value
```

For example, if the condition name is `root_actualdeletionperiod`, the condition value is 40 and the operator is `Equal`, the Boolean statement is:

```
<item>.root_actualdeletionperiod.value = = 40
```

The query returns all items whose `root_actualdeletionperiod` is 40, assuming there are no other conditions.

If the `conditionsLogicalOperator` argument is `AND`, the query returns the items that meet all conditions in the `conditions` collection. If `conditionsLogicalOperator` is `OR`, the query returns the items that meet at least one of the conditions in the `conditions` collection.

The following table lists the comparison operators:

Operator	Type of Condition/Comments
ChangedDuring	<p>Date</p> <p>This is a range check. The condition value is specified in hours. If the value of the date property lies in the range of the time the method is invoked plus or minus the condition value, the condition is true.</p> <p>For example, if the condition value is 24, the condition is true if the value of the date property is between yesterday at this time and tomorrow at this time.</p> <p>Note: The name <code>ChangedDuring</code> is kept to preserve backward compatibility. In previous versions, the operator was used only with create and modify time properties.</p>
Equal	String and numerical
EqualIgnoreCase	String
Greater	Numerical
GreaterEqual	Numerical
In	<p>String, numerical, and list</p> <p>The condition's value is a list. The condition is true if the value of the property is one of the values in the list.</p>
InList	<p>List</p> <p>The condition's value and the property's value are lists.</p> <p>The condition is true if all the values in the condition's list also appear in the item's property list. There can be more property values than specified in the condition without affecting the truth of the condition.</p>
IsNull	<p>String, numerical, and list</p> <p>The item's property has no value. When operator <code>IsNull</code> is used, the value of the condition is ignored, and in some cases can be nil.</p>
Less	Numerical
LessEqual	Numerical
Like	<p>String</p> <p>The condition's value is a substring of the value of the property's value. The condition's value must be bracketed with percentage signs (%). For example, <code>%Bi%</code> matches <code>Bismark</code> and <code>Bay of Biscay</code>, but not <code>biscuit</code>.</p>
LikeIgnoreCase	<p>String</p> <p>Use the <code>LikeIgnoreCase</code> operator as you use the <code>Like</code> operator. The match, however is not case-sensitive. Therefore, <code>%Bi%</code> matches <code>biscuit</code>.</p>

Operator	Type of Condition/Comments
NotEqual	String and numerical
UnchangedDuring	<p>Date</p> <p>This is a range check. The condition value is specified in hours. If the value of the date property is in the range of the time the method is invoked plus or minus the condition value, the condition is false. If it lies outside that range, the condition is true.</p> <p>For example, if the condition value is 24, the condition is true if the value of the date property is before yesterday at this time or after tomorrow at this time.</p> <p>Note: The name <code>UnchangedDuring</code> is kept to preserve backward compatibility. In previous versions, the operator was used only with create and modify time properties.</p>

Example of Setting Up a Condition:

```
FloatCondition fc = new FloatCondition();
FloatProp fp = new FloatProp();
fp.setName("attr_name");
fp.setValue(11);
fc.setCondition(fp);
fc.setFloatOperator(FloatCondition.floatOperatorEnum.Equal);
```

Example of Querying for Inherited Properties:

The target CI is `sample` which has two attributes, `name` and `size`. `sampleII` extends the CI with two attributes, `level` and `grade`. This example sets up a query for the properties of `sampleII` that were inherited from `sample` by specifying them by name.

```
GetFilteredCIsByType request = new GetFilteredCIsByType()
request.setCmdContext(cmdContext)
request.setType("sampleII")
CustomProperties customProperties = new CustomProperties();
PropertiesList propertiesList = new PropertiesList();
propertiesList.addPropertyName("name");
propertiesList.addPropertyName("size");
customProperties.setPropertiesList(propertiesList);
request.setProperties(customProperties)
```

Input

Parameter	Comment
cmdContext	For details, see "CmdContext" on page 272.

Parameter	Comment
type	The class name. For details, see "Type Name" on page 273 . The type can be any of the types defined using the CI Type Manager. For details, see CI Type Manager in the Modeling Guide.
properties	The data to be returned on each CI (called the Query Layout in the user interface). For details, see "CustomProperties" on page 266 .
conditions	A collection of name-value pairs and the operators that relate one to the other. For example, <code>host_hostname like QA</code> .
conditionsLogicalOperator	<ul style="list-style-type: none">• AND. All the conditions must be met.• OR. At least one of the conditions must be met.

Output

Parameter	Comment
CIs	Collection of CI elements.
chunkInfo	For details, see "ChunkInfo" on page 276 and "Processing Large Responses" on page 264 .

getQueryNameOfView

The `getQueryNameOfView` method retrieves the name of the TQL on which the specified view is based.

Input

Parameter	Comment
cmdbContext	For details, see "CmdbContext" on page 272 .
viewName	The name of a view, that is, a sub-set of the class model in the RTSM.

Output

Parameter	Comment
queryName	The name of the TQL in the RTSM on which the view is based.

getTopologyQueryExistingResultByName

The `getTopologyQueryExistingResultByName` method retrieves the most recent result of running the specified TQL. The call does not run the TQL. If there are no results from a previous run, nothing is returned.

Input

Parameter	Comment
cmdbContext	For details, see " CmdbContext " on page 272.
queryName	The name of a TQL.
queryTypedProperties	A collection of sets of properties to retrieve for items of a specific Configuration Item Type.

Output

Parameter	Comment
queryName	The name of the TQL in the RTSM on which the view is based.

getTopologyQueryResultCountByName

The `getTopologyQueryResultCountByName` method retrieves the number of instances of each node that matches the specified query.

Input

Parameter	Comment
cmdbContext	For details, see " CmdbContext " on page 272.
queryName	The name of a TQL.
countInvisible	If true, the output includes CIs defined as invisible in the query.

Output

Parameter	Comment
queryName	The name of the TQL in the RTSM on which the view is based.

pullTopologyMapChunks

The `pullTopologyMapChunks` method retrieves one of the chunks that contain the response to a method.

Each chunk contains a `topologyMap` element that is part of the response. The first chunk is numbered 1, so the retrieval loop counter iterates from 1 to `<response object>.getChunkInfo().getNumberOfChunks()`.

For details, see "[ChunkInfo](#)" on page 276 and "[Query the RTSM](#)" on page 264.

The client application must be able to handle the partial maps. See the following example of handling a CI collection and the example of merging chunks to a map in "[Query Example](#)" on page 296.

Input

Parameter	Comment
cmdbContext	For details, see " CmdbContext " on page 272.
ChunkRequest	The number of the chunk to retrieve and the <code>ChunkInfo</code> that is returned by the query method.

Output

Parameter	Comment
topologyMap	For details, see " TopologyMap " on page 275.
comments	For internal use only.

Example of Handling Chunks:

```

GetCIsByType request =
    new GetCIsByType(cmdbContext, typeName, customProperties);
GetCIsByTypeResponse response =
    ucmbService.getCIsByType(request);
ChunkRequest chunkRequest = new ChunkRequest();
chunkRequest.setChunkInfo(response.getChunkInfo());
for(int j=1; j<=response.getChunkInfo().getNumberOfChunks(); j++){
    chunkRequest.setChunkNumber(j);
    PullTopologyMapChunks req =new
        PullTopologyMapChunks(cmdbContext,chunkRequest);
    PullTopologyMapChunksResponse res =
        ucmbService.pullTopologyMapChunks(req);
    for(int m=0 ;
        m < res.getTopologyMap().getCINodes().sizeCINodeList()
;
        m++) {
        CIs cis =
            res.getTopologyMap().getCINodes().getCINode(m).getCIs
();
        for(int i=0 ; i < cis.sizeCICollection() ; i++) {
            // your code to process the CIs
        }
    }
}

GetCIsByType request =
    new GetCIsByType(cmdbContext, typeName, customProperties);
GetCIsByTypeResponse response =
    ucmbService.getCIsByType(request);
ChunkRequest chunkRequest = new ChunkRequest();
chunkRequest.setChunkInfo(response.getChunkInfo());
for(int j=1 ; j <= response.getChunkInfo().getNumberOfChunks() ;

```

```
j++) {  
    chunkRequest.setChunkNumber(j);  
    PullTopologyMapChunks req = new PullTopologyMapChunks  
(cmdbContext, chunkRequest);  
    PullTopologyMapChunksResponse res =  
        ucmdbService.pullTopologyMapChunks(req);  
    for(int m=0 ;  
        m < res.getTopologyMap().getCINodes().sizeCINodeList()  
    ;  
        m++) {  
        CIs cis =  
            res.getTopologyMap().getCINodes().getCINode(m).getCIs  
        ();  
        for(int i=0 ; i < cis.sizeCIIList() ; i++) {  
            // your code to process the CIs  
        }  
    }  
}
```

releaseChunks

The `releaseChunks` method frees the memory of the chunks that contain the data from the query.

Tip: The server discards the data after ten minutes. Calling this method to discard the data as soon as it has been read conserves server resources.

Input

Parameter	Comment
cmdbContext	For details, see " CmdbContext " on page 272.
chunksKey	The identifier of the data on the server that was chunked. The key is an element of <code>ChunkInfo</code> .

UCMDB Update Methods

This section provides information on the following methods:

- ["addCIsAndRelations" below](#)
- ["addCustomer" on the next page](#)
- ["deleteCIsAndRelations" on the next page](#)
- ["removeCustomer" on the next page](#)
- ["updateCIsAndRelations" on the next page](#)

addCIsAndRelations

The `addCIsAndRelations` method adds or updates CIs and relations.

If the CIs or relations do not exist in the RTSM, they are added and their properties are set according to the contents of the `CIsAndRelationsUpdates` argument.

If the CIs or relations do exist in the RTSM, they are updated with the new data, if `updateExisting` is **true**.

If `updateExisting` is **false**, `CIsAndRelationsUpdates` cannot reference existing configuration items or relations. Any attempt to reference existing items when `updateExisting` is false results in an exception.

If `updateExisting` is **true**, the add or update operation is performed without validating the CIs, regardless of the value of `ignoreValidation`.

If `updateExisting` is **false** and `ignoreValidation` is **true**, the add operation is performed without validating the CIs.

If `updateExisting` is **false** and `ignoreValidation` is **false**, the CIs are validated before the add operation.

Relations are never validated.

`CreatedIDsMap` is a map or dictionary of type `ClientIDToCmdbID` that connects the client's temporary IDs with the corresponding real RTSM IDs.

Input

Parameter	Comment
<code>cmdbContext</code>	For details, see "CmdbContext" on page 272 .
<code>updateExisting</code>	Set to <i>true</i> to update items that already exist in the RTSM. Set to <i>false</i> to throw an exception if any item already exists.
<code>CIsAndRelationsUpdates</code>	The items to update or create. For details, see "CIsAndRelationsUpdates" on page 267 .
<code>ignoreValidation</code>	If true, no check is performed before updating the RTSM.

Output

Parameter	Comment
CreatedIDsMap	The map of client IDs to RTSM IDs. For details, see the description above.
comments	For internal use only.

addCustomer

The `addCustomer` method adds a customer.

Input

Parameter	Comment
CustomerID	The numeric ID of the customer.

deleteCIsAndRelations

The `deleteCIsAndRelations` method removes the specified configuration items and relations from the RTSM.

When a CI is deleted and the CI is one end of one or more `Relation` items, those `Relation` items are also deleted.

Input

Parameter	Comment
cmdbContext	For details, see " CmdbContext " on page 272.
CIsAndRelationsUpdates	The items to delete. For details, see " CIsAndRelationsUpdates " on page 267

removeCustomer

The `removeCustomer` method deletes a customer record.

Input

Parameter	Comment
CustomerID	The numeric ID of the customer.

updateCIsAndRelations

The `updateCIsAndRelations` method updates the specified CIs and relations.

Update uses the property values from the `CIsAndRelationsUpdates` argument. If any of the CIs or relations do not exist in the RTSM, an exception is thrown.

`CreatedIDsMap` is a map or dictionary of type `ClientIDToCmdbID` that connects the client's temporary IDs with the corresponding real RTSM IDs.

Input

Parameter	Comment
<code>cmdbContext</code>	For details, see "CmdbContext" on page 272 .
<code>CIsAndRelationsUpdates</code>	The items to update. For details, see "CIsAndRelationsUpdates" on page 267 .
<code>ignoreValidation</code>	If true, no check is performed before updating the RTSM.

Output

Parameter	Comment
<code>CreatedIDsMap</code>	The map of client IDs to RTSM IDs. For details, see "addCIsAndRelations" on page 288 .

UCMDB Impact Analysis Methods

This section provides information on the following methods:

- "calculateImpact" below
- "getImpactPath" below
- "getImpactRulesByNamePrefix" on the next page

calculateImpact

The `calculateImpact` method calculates which CIs are affected by a given CI according to the rules defined in the RTSM.

This shows the effect of an event triggering of the rule. The `identifier` output of `calculateImpact` is used as input for "getImpactPath" below.

Input

Parameter	Comment
<code>cmdbContext</code>	For details, see "CmdbContext" on page 272.
<code>impactCategory</code>	The type of event that would trigger the rule being simulated.
<code>IDs</code>	A collection of ID elements.
<code>impactRulesNames</code>	A collection of <code>ImpactRuleName</code> elements.
<code>severity</code>	The severity of the triggering event.

Output

Parameter	Comment
<code>impactTopology</code>	For details, see "Topology" on page 275.
<code>identifier</code>	The key to the server response.

getImpactPath

The `getImpactPath` method retrieves the topology graph of the path between the affected CI and the CI that affects it.

The `identifier` output of "calculateImpact" above is used as the `identifier` input argument of `getImpactPath`.

Input

Parameter	Comment
<code>cmdbContext</code>	For details, see "CmdbContext" on page 272.

Parameter	Comment
identifier	The key to the server response that was returned by <code>calculateImpact</code> .
relation	A Relation based on one of the " ShallowRelation "s returned by <code>calculateImpact</code> in the <code>impactTopology</code> element.

Output

Parameter	Comment
impactPathTopology	A <code>CI</code> s collection and an <code>ImpactRelations</code> collection.
comments	For internal use only.

An `ImpactRelations` element consists of an `ID`, `type`, `end1ID`, `end2ID`, a rule, and an action.

getImpactRulesByNamePrefix

The `getImpactRulesByNamePrefix` method retrieves rules using a prefix filter.

This method applies to impact rules that are named with a prefix that indicates the context to which they apply, for example, `SAP_myrule`, `ORA_myrule`, and so on. This method filters all impact rule names for those beginning with the prefix specified by the `ruleNamePrefixFilter` argument.

Input

Parameter	Comment
cmdbContext	For details, see " CmdbContext " on page 272.
ruleNamePrefixFilter	A string containing the first letters of the rule names to match.

Output

Parameter	Comment
impactRules	<code>impactRules</code> is composed of zero or more <code>impactRule</code> . An <code>impactRule</code> , which specifies the effect of a change, is composed of <code>ruleName</code> , <code>description</code> , <code>queryName</code> , and <code>isActive</code> .

Use Cases

The following use cases assume two systems:

- BSM server
- A third-party system that contains a repository of configuration items

This section includes the following topics:

- ["Populating the RTSM" below](#)
- ["Querying the RTSM " below](#)
- ["Querying the Class Model" below](#)
- ["Analyzing Change Impact " below](#)

Populating the RTSM

Use cases:

- A third-party asset management updates the RTSM with information available only in asset management
- A number of third-party systems populate the RTSM to create a central RTSM that can track changes and perform impact analysis
- A third-party system creates Configuration Items and Relations according to third-party business logic to leverage the RTSM query capabilities

Querying the RTSM

Use cases:

- A third-party system gets the Configuration Items and Relations that represent the SAP system by getting the results of the SAP TQL
- A third-party system gets the list of Oracle servers that have been added or changed in the last five hours
- A third-party system gets the list of servers whose host name contains the substring *lab*
- A third-party system finds the elements related to a given CI by getting its neighbors

Querying the Class Model

Use cases:

- A third-party system enables users to specify the set of data to be retrieved from the RTSM. A user interface can be built over the class model to show users the possible properties and prompt them for required data. The user can then choose the information to be retrieved.
- A third-party system explores the class model when the user cannot access the BSM user interface.

Analyzing Change Impact

Use case:

A third-party system outputs a list of the business services that could be impacted by a change on a specified host.

Examples

This section includes the following topics:

- "The Example Base Class" below
- "Query Example" on the next page
- "Update Example" on page 308
- "Class Model Example" on page 312
- "Impact Analysis Example" on page 314
- "Adding Credentials Example" on page 316

The Example Base Class

```
package com.hp.ucmdb.demo;
import com.hp.ucmdb.generated.services.UcmdbService;
import com.hp.ucmdb.generated.services.UcmdbServiceStub;
import com.hp.ucmdb.generated.types.CmdbContext;
import org.apache.axis2.AxisFault;
import org.apache.axis2.transport.http.HTTPConstants;

import org.apache.axis2.transport.http.HttpTransportProperties;
import java.net.MalformedURLException;
import java.net.URL;

/**
 * User: hbarkai
 * Date: Jul 12, 2007
 */
abstract class Demo {

    UcmdbService stub;
    CmdbContext context;

    public void initDemo() {
        try {
            setStub(createUcmdbService("admin", "admin"));
            setContext();
        } catch (Exception e) {
            //handle exception
        }
    }

    public UcmdbService getStub() {
        return stub;
    }

    public void setStub(UcmdbService stub) {
        this.stub = stub;
    }
}
```

```

public CmdbContext getContext() {
    return context;
}

public void setContext() {
    CmdbContext context = new CmdbContext();
    context.setCallerApplication("demo");
    this.context = context;
}

//connection to service - for axis2/jibx client
private static final String PROTOCOL = "http";
private static final String HOST_NAME = "host_name";
private static final int PORT = 21212;
private static final String FILE = "/axis2/services/UcmdbContextService";
protected UcmdbContextService createUcmdbContextService
    (String username, String password) throws Exception{
    URL url;
    UcmdbContextServiceStub serviceStub;

    try {
        url = new URL
            (Demo.PROTOCOL, Demo.HOST_NAME,
             Demo.PORT, Demo.FILE);
        serviceStub = new UcmdbContextServiceStub(url.toString());
        HttpTransportProperties.Authenticator auth =
            new HttpTransportProperties.Authenticator();
        auth.setUsername(username);
        auth.setPassword(password);
        serviceStub._getServiceClient().getOptions().setProperty
            (HTTPConstants.AUTHENTICATE, auth);

    } catch (AxisFault axisFault) {
        throw new Exception
            ("Failed to create SOAP adapter for "
             + Demo.HOST_NAME , axisFault);

    } catch (MalformedURLException e) {
        throw new Exception
            ("Failed to create SOAP adapter for "
             + Demo.HOST_NAME, e);

    }
    return serviceStub;
}
}

```

Query Example

```

package com.hp.ucmdb.demo;
import com.hp.ucmdb.generated.params.query.*;
import com.hp.ucmdb.generated.services.UcmdbContextFaultException;
import com.hp.ucmdb.generated.services.UcmdbContextService;

```



```
import com.hp.ucmdb.generated.types.*;
import com.hp.ucmdb.generated.types.props.*;
import java.rmi.RemoteException;

public class QueryDemo extends Demo{
    UcmdbService stub;
    CmdbContext context;

    public void getCIsByTypeDemo() {
        GetCIsByType request = new GetCIsByType();
        //set cmdbcontext
        CmdbContext cmdbContext = getContext();
        request.setCmdbContext(cmdbContext);
        //set CIs type
        request.setType("anyType");
        //set CIs properties to be retrieved
        CustomProperties customProperties = new CustomProperties();
        PredefinedProperties predefinedProperties =
            new PredefinedProperties();
        SimplePredefinedProperty simplePredefinedProperty =
            new SimplePredefinedProperty();
        simplePredefinedProperty.setName
            (SimplePredefinedProperty.nameEnum.DERIVED);
        SimplePredefinedPropertyCollection
            simplePredefinedPropertyCollection =
                new SimplePredefinedPropertyCollection();

        simplePredefinedPropertyCollection.addSimplePredefinedProperty
            (simplePredefinedProperty);
        predefinedProperties.setSimplePredefinedProperties
            (simplePredefinedPropertyCollection);
        customProperties.setPredefinedProperties
            (predefinedProperties);
        request.setProperties(customProperties);
        try {
            GetCIsByTypeResponse response =
                getStub().getCIsByType(request);
            TopologyMap map =
                getTopologyMapResultFromCIs
                    (response.getCIs(), response.getChunkInfo());
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }
    }

    public void getCIsByIdDemo() {
        GetCIsById request = new GetCIsById();
        CmdbContext cmdbContext = getContext();
    }
```

```
//set cmdbcontext
request.setCmdbContext(cmdbContext);
//set ids
ID id1 = new ID();
id1.setBase("cmdbobjectidCIT1");
ID id2 = new ID();
id2.setBase("cmdbobjectidCIT2");
IDs ids = new IDs();
ids.addID(id1);
ids.addID(id2);
request.setIDs(ids);
//set CIs properties to be retrieved
TypedPropertiesCollection properties =
    new TypedPropertiesCollection();

TypedProperties typedProperties1 =
    new TypedProperties();
typedProperties1.setType("CIT1");

CustomTypedProperties customProperties1 =
    new CustomTypedProperties();
PredefinedTypedProperties predefinedProperties1 =
    new PredefinedTypedProperties();
SimpleTypedPredefinedProperty simplePredefinedProperty1 =
    new SimpleTypedPredefinedProperty();
simplePredefinedProperty1.setName
    (SimpleTypedPredefinedProperty.nameEnum.CONCRETE);
SimpleTypedPredefinedPropertyCollection
    simplePredefinedPropertyCollection1 =
        new SimpleTypedPredefinedPropertyCollection();
simplePredefinedPropertyCollection1
    .addSimpleTypedPredefinedProperty
        (simplePredefinedProperty1);

predefinedProperties1.
    setSimpleTypedPredefinedProperties
        (simplePredefinedPropertyCollection1);
customProperties1.
    setPredefinedTypedProperties
        (predefinedProperties1);
typedProperties1.setProperties(customProperties1);
properties.addTypedProperties(typedProperties1);

TypedProperties typedProperties2 =
    new TypedProperties();
typedProperties2.setType("CIT2");
CustomTypedProperties customProperties2 =
    new CustomTypedProperties();
PredefinedTypedProperties predefinedProperties2 =
    new PredefinedTypedProperties();
SimpleTypedPredefinedProperty simplePredefinedProperty2 =
    new SimpleTypedPredefinedProperty();
```

```
simplePredefinedProperty2.setName
    (SimpleTypedPredefinedProperty.nameEnum.NAMING);
SimpleTypedPredefinedPropertyCollection
    simplePredefinedPropertyCollection2 =
        new SimpleTypedPredefinedPropertyCollection();

simplePredefinedPropertyCollection2.
    addSimpleTypedPredefinedProperty
        (simplePredefinedProperty2);

predefinedProperties2.setSimpleTypedPredefinedProperties
    (simplePredefinedPropertyCollection2);
customProperties2.setPredefinedTypedProperties
    (predefinedProperties2);
typedProperties2.setProperties(customProperties2);
properties.addTypedProperties(typedProperties2);

request.setCIsTypedProperties(properties);
try {
    GetCIsByIdResponse response =
        getStub().getCIsById(request);
    CIs cis = response.getCIs();
} catch (RemoteException e) {
    //handle exception
} catch (UcmdbFaultException e) {
    //handle exception
}
}

public void getFilteredCIsByTypeDemo() {
    GetFilteredCIsByType request = new GetFilteredCIsByType();
    CmdbContext cmdbContext = getContext();
    //set cmdbcontext
    request.setCmdbContext(cmdbContext);
    //set CIs type
    request.setType("anyType");
    //sets Filter conditions
    Conditions conditions = new Conditions();
    IntConditions intConditions = new IntConditions();
    IntCondition intCondition = new IntCondition();
    IntProp intProp = new IntProp();
    intProp.setName("int_attr1");

    intProp.setValue(100);
    intCondition.setCondition(intProp);
    intCondition.setIntOperator
        (IntCondition.intOperatorEnum.Greater);
    intConditions.addIntCondition(intCondition);

    conditions.setIntConditions(intConditions);
    request.setConditions(conditions);
    //set logical operator for conditions
```

```
request.setConditionsLogicalOperator
    (GetFilteredCIsByType.conditionsLogicalOperatorEnum.AND);
//set CIs properties to be retrieved
CustomProperties customProperties =
    new CustomProperties();
PredefinedProperties predefinedProperties =
    new PredefinedProperties();
SimplePredefinedProperty simplePredefinedProperty =
    new SimplePredefinedProperty();
simplePredefinedProperty.setName
    (SimplePredefinedProperty.nameEnum.NAMING);

SimplePredefinedPropertyCollection
    simplePredefinedPropertyCollection =
        new SimplePredefinedPropertyCollection();
simplePredefinedPropertyCollection.
    addSimplePredefinedProperty
        (simplePredefinedProperty);
predefinedProperties.setSimplePredefinedProperties
    (simplePredefinedPropertyCollection);
customProperties.setPredefinedProperties
    (predefinedProperties);

request.setProperties(customProperties);
try {
    GetFilteredCIsByTypeResponse response =
        getStub().getFilteredCIsByType(request);
    TopologyMap map =
        getTopologyMapResultFromCIs
            (response.getCIs(), response.getChunkInfo());

} catch (RemoteException e) {
    //handle exception
} catch (UcldbFaultException e) {
    //handle exception
}
}

public void executeTopologyQueryByNameDemo() {
    ExecuteTopologyQueryByName request = new
ExecuteTopologyQueryByName();
    CmdbContext cmdbContext = getContext();
    //set cmdbcontext
    request.setCmdbContext(cmdbContext);
    //set query name
    request.setQueryName("queryName");

    try {
        ExecuteTopologyQueryByNameResponse response =
            getStub().executeTopologyQueryByName(request);
        TopologyMap map =
```

```

        getTopologyMapResult
            (response.getTopologyMap(), response.getChunkInfo
());
    } catch (RemoteException e) {
        //handle exception
    } catch (UcldbFaultException e) {
        //handle exception
    }
}

// assume the follow query was defined at UCMDB
// Query Name: exampleQuery
// Query sketch:
//
//                               Host
//                               /  \
//                               ip   Disk
// Query Parameters:
//      Host-
//          host_os (like)
//      Disk-
//          disk_failures (equal)

public void executeTopologyQueryByNameWithParametersDemo() {
    ExecuteTopologyQueryByNameWithParameters request =
        new ExecuteTopologyQueryByNameWithParameters();
    CmdbContext cmdbContext = getContext();
    //set cmdbcontext
    request.setCmdbContext(cmdbContext);
    //set query name
    request.setQueryName("queryName");
    //set parameters
    ParameterizedNode hostParametrizedNode =
        new ParameterizedNode();
    hostParametrizedNode.setNodeLabel("Host");
    CIProperties parameters = new CIProperties();
    StrProps strProps = new StrProps();
    StrProp strProp = new StrProp();
    strProp.setName("host_os");
    strProp.setValue("%2000%");
    strProps.addStrProp(strProp);
    parameters.setStrProps(strProps);
    hostParametrizedNode.setParameters(parameters);
    request.addParameterizedNodes(hostParametrizedNode);
    ParameterizedNode diskParametrizedNode =
        new ParameterizedNode();

    diskParametrizedNode.setNodeLabel("Disk");
    CIProperties parameters1 = new CIProperties();
    IntProps intProps = new IntProps();

```

```

    IntProp intProp = new IntProp();
    intProp.setName("disk_failures");
    intProp.setValue(30);
    intProps.addIntProp(intProp);
    parameters1.setIntProps(intProps);
    diskParametrizedNode.setParameters(parameters1);

    request.addParameterizedNodes(diskParametrizedNode);
    try {
        ExecuteTopologyQueryByNameWithParametersResponse
            response =
                getStub().executeTopologyQueryByNameWithParameters
                    (request);
        TopologyMap map =
            getTopologyMapResult
                (response.getTopologyMap(), response.getChunkInfo
                    ());
    } catch (RemoteException e) {
        //handle exception
    } catch (UcmdbFaultException e) {
        //handle exception
    }
}

/    // assume the follow query was defined at UCMDB
// Query Name: exampleQuery
// Query sketch:
//
//                                     Host
//                                     /  \
//                                     ip   Disk
// Query Parameters:
//     Host-
//         host_os (like)
//     Disk-
//         disk_failures (equal)

public void executeTopologyQueryWithParametersDemo() {
    ExecuteTopologyQueryWithParameters request =
        new ExecuteTopologyQueryWithParameters();
    CmdbContext cmdbContext = getContext();
    //set cmdbcontext
    request.setCmdbContext(cmdbContext);
    //set query definition
    String queryXml = "<xml that represents the query above>";
    request.setQueryXml(queryXml);
    //set parameters
    ParameterizedNode hostParametrizedNode =
        new ParameterizedNode();

```

```

hostParametrizedNode.setNodeLabel("Host");
    CIProperties parameters = new CIProperties();
    StrProps strProps = new StrProps();
    StrProp strProp = new StrProp();
    strProp.setName("host_os");
    strProp.setValue("%2000%");
    strProps.addStrProp(strProp);
    parameters.setStrProps(strProps);
    hostParametrizedNode.setParameters(parameters);
    request.addParameterizedNodes(hostParametrizedNode);
    ParameterizedNode diskParametrizedNode =
        new ParameterizedNode();
    diskParametrizedNode.setNodeLabel("Disk");
    CIProperties parameters1 = new CIProperties();
    IntProps intProps = new IntProps();
    IntProp intProp = new IntProp();
    intProp.setName("disk_failures");
    intProp.setValue(30);
    intProps.addIntProp(intProp);
    parameters1.setIntProps(intProps);
    diskParametrizedNode.setParameters(parameters1);
    request.addParameterizedNodes(diskParametrizedNode);

    try {
        ExecuteTopologyQueryWithParametersResponse
        response = getStub().executeTopologyQueryWithParameters
            (request);
        TopologyMap map =
            getTopologyMapResult
                (response.getTopologyMap(), response.getChunkInfo
());
    } catch (RemoteException e) {
        //handle exception
    } catch (UcmdbFaultException e) {
        //handle exception
    }
}

public void getCINeighboursDemo() {
    GetCINeighbours request = new GetCINeighbours();
    //set cmdbcontext
    CmdbContext cmdbContext = getContext();
    request.setCmdbContext(cmdbContext);
    // set CI id
    ID id = new ID();
    id.setBase("cmdbobjectidCIT1");
    request.setID(id);
    //set neighbour type
    request.setNeighbourType("neighbourType");
}

```

```
//set Neighbours CIs propeties to be retrieved
TypedPropertiesCollection properties =
    new TypedPropertiesCollection();
TypedProperties typedProperties1 = new TypedProperties();
typedProperties1.setType("neighbourType");
CustomTypedProperties customProperties1 =
    new CustomTypedProperties();
PredefinedTypedProperties predefinedProperties1 =
    new PredefinedTypedProperties();

QualifierProperties qualifierProperties =
    new QualifierProperties();
qualifierProperties.addQualifierName("ID_ATTRIBUTE");
predefinedProperties1.setQualifierProperties
(qualifierProperties);
customProperties1.setPredefinedTypedProperties
(predefinedProperties1);
typedProperties1.setProperties(customProperties1);
properties.addTypedProperties(typedProperties1);
request.setCIProperties(properties);

TypedPropertiesCollection relationsProperties =
    new TypedPropertiesCollection();
TypedProperties typedProperties2 = new TypedProperties();
typedProperties2.setType("relationType");
CustomTypedProperties customProperties2 =
    new CustomTypedProperties();

PredefinedTypedProperties predefinedProperties2 =
    new PredefinedTypedProperties();
SimpleTypedPredefinedProperty simplePredefinedProperty2 =
    new SimpleTypedPredefinedProperty();
simplePredefinedProperty2.setName

    (SimpleTypedPredefinedProperty.nameEnum.CONCRETE);
SimpleTypedPredefinedPropertyCollection
    simplePredefinedPropertyCollection2 =
        new SimpleTypedPredefinedPropertyCollection();
simplePredefinedPropertyCollection2.
    addSimpleTypedPredefinedProperty
        (simplePredefinedProperty2);
predefinedProperties2.
    setSimpleTypedPredefinedProperties
        (simplePredefinedPropertyCollection2);
customProperties2.setPredefinedTypedProperties
    (predefinedProperties2);
typedProperties2.setProperties(customProperties2);
relationsProperties.addTypedProperties(typedProperties2);
request.setRelationProperties(relationsProperties);

try {
    GetCINeighboursResponse response =
```



```

        getStub().getCINeighbours(request);
        Topology topology = response.getTopology();
    } catch (RemoteException e) {
        //handle exception
    } catch (UcldbFaultException e) {
        //handle exception
    }
}

//get Topology Map for chunked/non-chunked result
private TopologyMap getTopologyMapResult(TopologyMap topologyMap,
ChunkInfo chunkInfo) {
    if(chunkInfo.getNumberOfChunks() == 0) {
        return topologyMap;
    } else {

        topologyMap = new TopologyMap();
        for(int i=1 ; i <= chunkInfo.getNumberOfChunks() ; i++) {
            ChunkRequest chunkRequest = new ChunkRequest();
            chunkRequest.setChunkInfo(chunkInfo);
            chunkRequest.setChunkNumber(i);
            PullTopologyMapChunks req =
                new PullTopologyMapChunks();
            req.setChunkRequest(chunkRequest);
            req.setCmdbContext(getContext());
            PullTopologyMapChunksResponse res = null;

            try {
                res = getStub().pullTopologyMapChunks(req);
                TopologyMap map = res.getTopologyMap();
                topologyMap = mergeMaps(topologyMap, map);
            } catch (RemoteException e) {
                //handle exception
            } catch (UcldbFaultException e) {
                //handle exception
            }
        }
    }
    return topologyMap;
}

private TopologyMap getTopologyMapResultFromCIs(CIs cis, ChunkInfo
chunkInfo) {
    TopologyMap topologyMap = new TopologyMap();
    if(chunkInfo.getNumberOfChunks() == 0) {
        CINode ciNode = new CINode();
        ciNode.setLabel("");
        ciNode.setCIs(cis);
        CINodes ciNodes = new CINodes();
        ciNodes.addCINode(ciNode);
    }
}

```

```

        topologyMap.setCINodes(ciNodes);
    } else {

        for(int i=1 ; i <= chunkInfo.getNumberOfChunks() ; i++) {
            ChunkRequest chunkRequest =
                new ChunkRequest();
            chunkRequest.setChunkInfo(chunkInfo);
            chunkRequest.setChunkNumber(i);
            PullTopologyMapChunks req =
                new PullTopologyMapChunks();
            req.setChunkRequest(chunkRequest);
            req.setCmdbContext(getContext());
            PullTopologyMapChunksResponse res = null;

            try {
                res = getStub().pullTopologyMapChunks(req);
            } catch (RemoteException e) {
                //handle exception
            } catch (UcmdbFaultException e) {
                //handle exception
            }
            TopologyMap map = res.getTopologyMap();
            topologyMap = mergeMaps(topologyMap, map);
        }

        //release chunks
        ReleaseChunks req = new ReleaseChunks();
        req.setChunksKey(chunkInfo.getChunksKey());
        req.setCmdbContext(getContext());

        try {
            getStub().releaseChunks(req);
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }
    }
    return topologyMap;
}

//=====
/*  WARNING merge will be correct only if a each node is given
    a unique name. This applies to both CI and Relation nodes .*/
//=====
private TopologyMap mergeMaps(TopologyMap topologyMap, TopologyMap
newMap) {
    for(int i=0 ; i < newMap.getCINodes().sizeCINodeList() ; i++ )
    {
        CINode ciNode = newMap.getCINodes().getCINode(i);
        boolean alreadyExist = false;
        if(topologyMap.getCINodes() == null) {

```

```

        topologyMap.setCINodes(new CINodes());
    }

    for(int j=0 ; j < topologyMap.getCINodes().sizeCINodeList
() ; j++) {
        CInode ciNode2 = topologyMap.getCINodes().getCInode
(j);
        if(ciNode2.getLabel().equals(ciNode.getLabel())){
            CIs cisTOAdd = ciNode.getCIs();
            CIs cis =
                mergeCIsGroups
                (topologyMap.getCINodes().getCInode(j).getCIs
(),
                    cisTOAdd);
            topologyMap.getCINodes().getCInode(j).setCIs(cis);
            alreadyExist = true;
        }
    }
    if(!alreadyExist) {
        topologyMap.getCINodes().addCInode(ciNode);
    }
}

for(int i=0 ; i < newMap.getRelationNodes
().sizeRelationNodeList() ; i++ ) {
    RelationNode relationNode =
        newMap.getRelationNodes().getRelationNode(i);
    boolean alreadyExist = false;
    if(topologyMap.getRelationNodes() == null) {
        topologyMap.setRelationNodes(new RelationNodes());
    }

    for(int j=0 ;
        j < topologyMap.getRelationNodes
().sizeRelationNodeList() ;
        j++) {
        RelationNode relationNode2 =
            topologyMap.getRelationNodes().getRelationNode(j);
        if(relationNode2.getLabel().equals
(relationNode.getLabel())){
            Relations relationsTOAdd =
relationNode.getRelations();
            Relations relations =
                mergeRelationsGroups
                (topologyMap.getRelationNodes().
                    getRelationNode(j).getRelations(),
                    relationsTOAdd);
            topologyMap.getRelationNodes().
                getRelationNode(j).setRelations(relations);
            alreadyExist = true;

```

```

        }
    }

    if(!alreadyExist) {
        topologyMap.getRelationNodes().addRelationNode
(relationNode);
    }
}

return topologyMap;
}

private Relations mergeRelationsGroups(Relations relations1,
Relations relations2) {
    for(int i=0 ; i < relations2.sizeRelationList() ; i++) {
        relations1.addRelation(relations2.getRelation(i));
    }
    return relations1;
}

private CIs mergeCIsGroups(CIs cis1, CIs cis2) {
    for(int i=0 ; i < cis2.sizeCICollection() ; i++) {
        cis1.addCI(cis2.getCI(i));
    }
    return cis1;
}
}

```

Update Example

```

import com.hp.ucmdb.generated.params.update.AddCIsAndRelations;

import
com.hp.ucmdb.generated.params.update.AddCIsAndRelationsResponse;

import com.hp.ucmdb.generated.params.update.UpdateCIsAndRelations;
import com.hp.ucmdb.generated.params.update.DeleteCIsAndRelations;
import com.hp.ucmdb.generated.services.UcmdbFault;
import com.hp.ucmdb.generated.types.*;
import com.hp.ucmdb.generated.types.update.CIsAndRelationsUpdates;
import com.hp.ucmdb.generated.types.update.ClientIDToCmdbID;
import java.rmi.RemoteException;
import java.util.ArrayList;
import java.util.List;

public class UpdateDemo extends Demo{

    public void getAddCIsAndRelationsDemo() {
        AddCIsAndRelations request = new AddCIsAndRelations();
        request.setCmdbContext(getContext());
    }
}

```

```
request.setUpdateExisting(true);

CIsAndRelationsUpdates updates = new CIsAndRelationsUpdates();

CIs cis = new CIs();

List<CI> listCI = new ArrayList<CI>();

CI ci = new CI();

ID id = new ID();

id.setString("templ");

id.setTemp(true);

ci.setID(id);

ci.setType("host");

CIProperties props = new CIProperties();

StrProps strProps = new StrProps();

StrProp strProp = new StrProp();

strProp.setName("host_key");

String value = "blabla";

strProp.setValue(value);

strProps.getStrProps().add(strProp);

props.setStrProps(strProps);

ci.setProps(props);

listCI.add(ci);

cis.setCIs(listCI);

updates.setCIsForUpdate(cis);

request.setCIsAndRelationsUpdates(updates);

try {

    AddCIsAndRelationsResponse response = getStub().addCIsAndRelations
(request);

    for(int i = 0 ; i < response.getCreatedIDsMaps().size() ; i++) {

        ClientIDToCmdbID idsMap = response.getCreatedIDsMaps().get(i);

        //do something

    }

} catch (RemoteException e) {

    //handle exception

} catch (UcmdbFault e) {

    //handle exception

}
```

```
    }  
}  
  
public void getUpdateCIsAndRelationsDemo() {  
    UpdateCIsAndRelations request = new UpdateCIsAndRelations();  
    request.setCmdbContext(getContext());  
    CIsAndRelationsUpdates updates = new CIsAndRelationsUpdates();  
    CIs cis = new CIs();  
    List<CI> listCI = new ArrayList<CI>();  
    CI ci = new CI();  
    ID id = new ID();  
    id.setString("templ");  
    id.setTemp(true);  
    ci.setID(id);  
    ci.setType("host");  
    CIProperties props = new CIProperties();  
    StrProps strProps = new StrProps();  
    StrProp hostKeyProp = new StrProp();  
    hostKeyProp.setName("host_key");  
    String hostKeyValue = "blabla";  
    hostKeyProp.setValue(hostKeyValue);  
    strProps.getStrProps().add(hostKeyProp);  
    StrProp hostOSProp = new StrProp();  
    hostOSProp.setName("host_os");  
    String hostOSValue = "winXP";  
    hostOSProp.setValue(hostOSValue);  
    strProps.getStrProps().add(hostOSProp);  
    StrProp hostDNSProp = new StrProp();  
    hostDNSProp.setName("host_dnsname");  
    String hostDNSValue = "dnsname";  
    hostDNSProp.setValue(hostDNSValue);  
    strProps.getStrProps().add(hostDNSProp);  
    props.setStrProps(strProps);  
    ci.setProps(props);  
}
```

```
listCI.add(ci);
cis.setCIs(listCI);
updates.setCIsForUpdate(cis);
request.setCIsAndRelationsUpdates(updates);
try {
    getStub().updateCIsAndRelations(request);
} catch (RemoteException e) {
    //handle exception
} catch (UcmdbFault e) {
    //handle exception
}
}

public void getDeleteCIsAndRelationsDemo() {
    DeleteCIsAndRelations request = new DeleteCIsAndRelations();
    request.setCmdbContext(getContext());
    CIsAndRelationsUpdates updates = new CIsAndRelationsUpdates();
    CIs cis = new CIs();
    List<CI> listCI = new ArrayList<CI>();
    CI ci = new CI();
    ID id = new ID();
    id.setString("stam");
    id.setTemp(true);
    ci.setID(id);
    ci.setType("host");
    CIProperties props = new CIProperties();
    StrProps strProps = new StrProps();
    StrProp strProp1 = new StrProp();
    strProp1.setName("host_key");
    String value1 = "for_delete";
    strProp1.setValue(value1);
    strProps.getStrProps().add(strProp1);
    props.setStrProps(strProps);
    ci.setProps(props);
}
```

```

        listCI.add(ci);
        cis.setCIs(listCI);
        updates.setCIsForUpdate(cis);
        request.setCIsAndRelationsUpdates(updates);
        try {
            getStub().deleteCIsAndRelations(request);
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFault e) {
            //handle exception
        }
    }
}

public static void main(String[] args) {
    try{
        UpdateDemo demo = new UpdateDemo();
        demo.initDemo();
        demo.getAddCIsAndRelationsDemo();
    } catch (Exception e) {
        System.out.println(e.getMessage());
        e.printStackTrace();
    }
}
}

```

Class Model Example

```

package com.hp.ucmdb.demo;
import com.hp.ucmdb.generated.params.classmodel.*;
import com.hp.ucmdb.generated.services.UcmdbFaultException;
import
com.hp.ucmdb.generated.types.classmodel.UcmdbClassModelHierarchy;
import com.hp.ucmdb.generated.types.classmodel.UcmdbClass;
import java.rmi.RemoteException;

public class ClassmodelDemo extends Demo{

    public void getClassAncestorsDemo() {

```



```
        GetClassAncestors request =
            new GetClassAncestors();
        request.setCmdbContext(getContext());
        request.setClassName("className");

        try {
            GetClassAncestorsResponse response =
                getStub().getClassAncestors(request);
            UcmdbClassModelHierarchy hierarchy =
                response.getClassHierarchy();
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }
    }

    public void getAllClassesHierarchyDemo() {
        GetAllClassesHierarchy request =
            new GetAllClassesHierarchy();
        request.setCmdbContext(getContext());
        try {
            GetAllClassesHierarchyResponse response =
                getStub().getAllClassesHierarchy(request);
            UcmdbClassModelHierarchy hierarchy =
                response.getClassesHierarchy();
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }
    }

    public void getCmdbClassDefinitionDemo() {
        GetCmdbClassDefinition request =
            new GetCmdbClassDefinition();
        request.setCmdbContext(getContext());
        request.setClassName("className");

        try {
            GetCmdbClassDefinitionResponse response =
                getStub().getCmdbClassDefinition(request);
            UcmdbClass ucmdbClass = response.getUcmdbClass();
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }
    }
}
```

Impact Analysis Example

```
package com.hp.ucmdb.demo;
import com.hp.ucmdb.generated.params.impact.*;
import com.hp.ucmdb.generated.services.UcmdbFaultException;
import com.hp.ucmdb.generated.types.*;
import com.hp.ucmdb.generated.types.impact.*;
import java.rmi.RemoteException;

/**
 * Date: Jul 17, 2007
 */
public class ImpactDemo extends Demo{

    //Impact Rule Name : impactExample
    //Impact Query:
    //          Network
    //          |
    //          Host
    //          |
    //          IP
    //Impact Action: network affect on ip ;severity 100% ; category:
    change
    //
    public void calculateImpactAndGetImpactPathDemo() {
        CalculateImpact request = new CalculateImpact();
        request.setCmdbContext(getContext());
        //set root cause ids
        IDs ids = new IDs();
        ID id = new ID();
        id.setBase("rootCauseCmdbID");
        ids.addID(id);

        request.setIDs(ids);
        //set impact category
        request.setImpactCategory("change");
        //set rule Names
        ImpactRuleNames impactRuleNames = new ImpactRuleNames();
        ImpactRuleName impactRuleName = new ImpactRuleName();
        impactRuleName.setBase("impactExample");
        impactRuleNames.addImpactRuleName(impactRuleName);
        request.setImpactRuleNames(impactRuleNames);
        //set severity
        request.setSeverity(100);
        CalculateImpactResponse response =
            new CalculateImpactResponse();

        request.setIDs(ids);
        //set impact category
        request.setImpactCategory("change");
        //set rule Names
```

```
ImpactRuleNames impactRuleNames = new ImpactRuleNames();
ImpactRuleName impactRuleName = new ImpactRuleName();
impactRuleName.setBase("impactExample");
impactRuleNames.addImpactRuleName(impactRuleName);
request.setImpactRuleNames(impactRuleNames);
//set severity
request.setSeverity(100);
CalculateImpactResponse response =
    new CalculateImpactResponse();

try {
    response = getStub().calculateImpact(request);
} catch (RemoteException e) {
    //handle exception
} catch (UcmdbFaultException e) {
    //handle exception
}

Identifier identifier= response.getIdentifier();
Topology topology = response.getImpactTopology();
Relation relation = topology.getRelations().getRelation(0);
GetImpactPath request2 = new GetImpactPath();
//set cmdb context
request2.setCmdbContext(getContext());
//set impact identifier
request2.setIdentifier(identifier);
//set shallowRelation
ShallowRelation shallowRelation = new ShallowRelation();
shallowRelation.setID(relation.getID());
shallowRelation.setEnd1ID(relation.getEnd1ID());
shallowRelation.setEnd2ID(relation.getEnd2ID());
shallowRelation.setType(relation.getType());
request2.setRelation(shallowRelation);

try {
    GetImpactPathResponse response2 =
        getStub().getImpactPath(request2);
    ImpactTopology impactTopology =
        response2.getImpactPathTopology();
} catch (RemoteException e) {
    //To change body of catch statement
    // use File | Settings | File Templates.
    e.printStackTrace();
} catch (UcmdbFaultException e) {
    //To change body of catch statement
    // use File | Settings | File Templates.
    e.printStackTrace();
}

}

public void getImpactRulesByGroupName() {
```

```

    GetImpactRulesByGroupName request =
        new GetImpactRulesByGroupName();
    //set cmdb context
    request.setCmdbContext(getContext());
    //set group names list
    request.addRuleGroupNameFilter("groupName1");
    request.addRuleGroupNameFilter("groupName2");

    try {
        GetImpactRulesByGroupNameResponse response =
            getStub().getImpactRulesByGroupName(request);
        ImpactRules impactRules = response.getImpactRules();
    } catch (RemoteException e) {
        //handle exception
    } catch (UcmdbFaultException e) {
        //handle exception
    }
}

public void getImpactRulesByNamePrefix() {
    GetImpactRulesByNamePrefix request =
        new GetImpactRulesByNamePrefix();
    //set cmdb context
    request.setCmdbContext(getContext());
    //set prefixes list
    request.addRuleNamePrefixFilter("prefix1");

    try {
        GetImpactRulesByNamePrefixResponse response =
            getStub().getImpactRulesByNamePrefix(request);
        ImpactRules impactRules = response.getImpactRules();
    } catch (RemoteException e) {
        //handle exception
    } catch (UcmdbFaultException e) {
        //handle exception
    }
}
}

```

Adding Credentials Example

```

import java.net.URL;
import org.apache.axis2.transport.http.HTTPConstants;
import org.apache.axis2.transport.http.HttpTransportProperties;
import com.hp.ucmdb.generated.params.discovery.*;
import com.hp.ucmdb.generated.services.DiscoveryService;
import com.hp.ucmdb.generated.services.DiscoveryServiceStub;
import com.hp.ucmdb.generated.types.BytesProp;
import com.hp.ucmdb.generated.types.BytesProps;
import com.hp.ucmdb.generated.types.CIProperties;

```

```
import com.hp.ucmdb.generated.types.CmdbContext;
import com.hp.ucmdb.generated.types.StrList;
import com.hp.ucmdb.generated.types.StrProp;
import com.hp.ucmdb.generated.types.StrProps;

public class test {
    static final String HOST_NAME = "hostname";
    static final int PORT = 808021212;
    private static final String PROTOCOL = "http";
    private static final String FILE =
"/axis2/services/DiscoveryService";

    private static final String PASSWORD = "admin";
    private static final String USERNAME = "admin";

    private static CmdbContext cmdbContext = new CmdbContext("ws
tests");

    public static void main(String[] args) throws Exception {
        // Get the stub object
        DiscoveryService discoveryService = getDiscoveryService();

        // Activate Job
        discoveryService.activateJob(new ActivateJobRequest("Range IPs
by ICMP", cmdbContext));

        // Get domain & probes info
        getProbesInfo(discoveryService);
        // Add credentilas entry for ntcmd protocol
        addNTCMDCredentialsEntry();
    }

    public static void addNTCMDCredentialsEntry() throws Exception {
        DiscoveryService discoveryService = getDiscoveryService();

        // Get domain name
        StrList domains =
            discoveryService.getDomainsNames(new
GetDomainsNamesRequest(cmdbContext)).getDomainNames();
        if (domains.sizeStrValueList() == 0) {
            System.out.println("No domains were found, can't create
credentials");
            return;
        }
        String domainName = domains.getStrValue(0);
        // Create propeties with one byte param
        CIProperties newCredsProperties = new CIProperties();

        // Add password property - this is of type bytes
        newCredsProperties.setBytesProps(new BytesProps());
        setPasswordProperty(newCredsProperties);
    }
}
```

```

        // Add user & domain properties - these are of type string
        newCredsProperties.setStrProps(new StrProps());
        setStringProperties("protocol_username", "test user",
newCredsProperties);
        setStringProperties("ntadminprotocol_ntdomain", "test doamin",
newCredsProperties);

        // Add new credentials entry
        discoveryService.addCredentialsEntry(new
AddCredentialsEntryRequest(domainName, "ntadminprotocol",
newCredsProperties, cmdbContext));
        System.out.println("new credentials craeted for domain: " +
domainName + " in ntcmd protocol");
    }

    private static void setPasswordProperty(CIProperties
newCredsProperties) {
        BytesProp bProp = new BytesProp();
        bProp.setName("protocol_password");
        bProp.setValue(new byte[] {101,103,102,104});
        newCredsProperties.getBytesProps().addBytesProp(bProp);
    }

    private static void setStringProperties(String propertyName,
String value, CIProperties newCredsProperties) {
        StrProp strProp = new StrProp();
        strProp.setName(propertyName);
        strProp.setValue(value);
        newCredsProperties.getStrProps().addStrProp(strProp);
    }

    private static void getProbesInfo(DiscoveryService
discoveryService) throws Exception {
        GetDomainsNamesResponse result =
discoveryService.getDomainsNames(new GetDomainsNamesRequest
(cmdbContext));
        // Go over all the domains
        if (result.getDomainNames().sizeStrValueList() > 0) {
            String domainName = result.getDomainNames().getStrValue
(0);

            GetProbesNamesResponse probesResult =
                discoveryService.getProbesNames(new
GetProbesNamesRequest(domainName, cmdbContext));
            // Go over all the probes
            for (int i=0; i<probesResult.getProbesNames
().sizeStrValueList(); i++) {
                String probeName = probesResult.getProbesNames
().getStrValue(i);
                // Check if connected
                IsProbeConnectedResponse connectedRequest =

```

```
        discoveryService.isProbeConnected(new
IsProbeConnectedRequest(domainName, probeName, cmdbContext));
        Boolean isConnected = connectedRequest.getIsConnected
();
        // Do something ...
        System.out.println("probe " + probeName + "
isconnect=" + isConnected);
    }
}

private static DiscoveryService getDiscoveryService() throws
Exception {
    DiscoveryService discoveryService = null;
    try {
        // Create service
        URL url = new URL(PROTOCOL,HOST_NAME,PORT, FILE);
        DiscoveryServiceStub serviceStub = new
DiscoveryServiceStub(url.toString());

        // Authenticate info
        HttpTransportProperties.Authenticator auth = new
HttpTransportProperties.Authenticator();
        auth.setUsername(USERNAME);
        auth.setPassword(PASSWORD);
        serviceStub._getServiceClient().getOptions().setProperty
(HTTPConstants.AUTHENTICATE,auth);

        discoveryService = serviceStub;
    } catch (Exception e) {
        throw new Exception("cannot create a connection to service
", e);
    }
    return discoveryService;
}
} // End class
```

Chapter 12

Internal-Global ID Conversion API

This chapter includes:

- Using the Internal-Global ID Conversion API 321
- How to Convert RTSM Internal IDs to Global IDs 322
- How to Convert Global IDs to RTSM Internal IDs 325

Using the Internal-Global ID Conversion API

BSM APIs work with internal CI IDs, while other tools integrated through a UCMDB work with global CI IDs. To use BSM APIs when you have global CI IDs, you must first convert them to internal CI IDs. Likewise, to send input from BSM to external software that integrates through a UCMDB, the internal CI IDs must first be converted to global CI IDs.

The Internal-Global ID Conversion API contains two methods, to convert RTSM internal CI IDs to global CI IDs, and to convert global CI IDs to RTSM internal CI IDs.

The log file for this API is located on the Gateway Server, at **<HPBSM root directory>\log\EJBContainer\bsm_sdk_utils.log**

How to Convert RTSM Internal IDs to Global IDs

To convert a given internal CI ID to a global CI ID, use the following URL:

**`http://<bsm.server.example>/topaz/bsmservices/customers/{customerId}
/rtsm/convertRtسمToGlobalIds?ciIds={cilds}`**

Parameters

- customerId - customer ID (use 1 for non-SaaS deployment)
- cilds - comma separated CI IDs

If the CI ID does not exist, it will not appear in the result map.

If no global ID is defined for a CI, 32 '0' characters are retrieved as the global ID mapped to the internal ID.

Example

HTTP Command

```
http://bsm.server.example/topaz/bsmservices/customers/1/rtsm/  
convertRtسمToGlobalIds?ciIds=35845d64ba6250d4d45d0e75e3e59001,  
316e438afc58135f9e87113bd4427d37,175fe30654e9a9dd16ddab66b2237c92,  
cb8a27f8ce900aa380c41ac1c74d8153,8066fe1fcb712926fe3793a75b0e733c,  
bbbc37f7ce900aa380c41ac1c74d8153
```

Result map

```
<map>
  <entry>
    <key>8066fe1fcb712926fe3793a75b0e733c</key>
    <value>00000000000000000000000000000000</value>
  </entry>
  <entry>
    <key>cb8a27f8ce900aa380c41ac1c74d8153</key>
    <value>cb8a27f8ce900aa380c41ac1c74d8153</value>
  </entry>
  <entry>
    <key>175fe30654e9a9dd16ddab66b2237c92</key>
    <value>175fe30654e9a9dd16ddab66b2237c92</value>
  </entry>
  <entry>
    <key>316e438afc58135f9e87113bd4427d37</key>
    <value>316e438afc58135f9e87113bd4427d37</value>
  </entry>
  <entry>
    <key>35845d64ba6250d4d45d0e75e3e59001</key>
    <value>00000000000000000000000000000000</value>
  </entry>
</map>
```

In this example, there are two CIs that have no global ID and will be mapped to 00000000000000000000000000000000.

Since there is no CI with ID that matches the last passed ID, it does not appear in the result map.

Error handling

Name	Error Code	Reason
BAD_REQUEST	400	Missing CI IDs or wrong input for cilds parameter. Number of passed CI IDs reaches the limit, which is 10000.

Name	Error Code	Reason
INTERNAL_SERVER_ERROR	500	General failure to run query against RTSM

How to Convert Global IDs to RTSM Internal IDs

To convert a given global CI ID to an internal CI ID, use the following URL:

**`http://<bsm.server.example>/topaz/bsmservices/customers/{customerId}
/rtsm/convertGlobalToRtsmIds?cilds={cilds}`**

Parameters

- `customerId` - customer ID (use 1 for non-SaaS deployment)
- `cilds` - comma separated CI IDs

If no CI with the given global ID appears in the RTSM model, no entry will appear in the result map.

Example

HTTP Command

```
http://bsm.server.example/topaz/bsmservices/customers/1/rtsm/convertGlobalToRtsmIds?ciIds=35845d64ba6250d4d45d0e75e3e59001,316e438afc58135f9e87113bd4427d37,175fe30654e9a9dd16ddab66b2237c92,cb8a27f8ce900aa380c41ac1c74d8153
```

Result map

```
<map>
  <entry>
    <key>316e438afc58135f9e87113bd4427d37</key>
    <value>316e438afc58135f9e87113bd4427d37</value>
  </entry>
  <entry>
    <key>175fe30654e9a9dd16ddab66b2237c92</key>
    <value>175fe30654e9a9dd16ddab66b2237c92</value>
  </entry>
  <entry>
    <key>cb8a27f8ce900aa380c41ac1c74d8153</key>
    <value>cb8a27f8ce900aa380c41ac1c74d8153</value>
  </entry>
</map>
```

Since there is no CI with a global ID that matches the first ID, it does not appear in the result map.

Error handling

Name	Error Code	Reason
BAD_REQUEST	400	Missing CI IDs or wrong input for cilds parameter. Number of passed CI IDs reaches the limit, which is 10000.
INTERNAL_SERVER_ERROR	500	General failure to run query against RTSM

Chapter 13

Data Flow Management API

This chapter includes:

- Data Flow Management API Overview328
- Conventions 329
- Data Flow Management Web Service 330
- Call the Web Service331
- Data Flow Management Methods332
- Code Sample 342

Data Flow Management API Overview

This chapter explains how third-party or custom tools can use the HP Data Flow Management Web Service to manage Data Flow Management.

For full documentation on the available operations, see the *HP Discovery and Dependency Mapping Schema Reference*. These files are located in the following folder:

**\\<BSM root directory>\AppServer\webapps\site.war\amdocs\eng\API_docs\DDM_Schema
\webframe.html**

Conventions

This chapter uses the following conventions:

- This style `Element` indicates that an item is an entity in the database or an element defined in the schema, including structures passed to or returned by methods. Plain text indicates that the item is being discussed in a general context.
- Data Flow Management elements and method arguments are spelled in the case in which they are specified in the schema. This usually means that a class name or generic reference to an instance of the class is capitalized. An element or argument to a method is not capitalized. For example, a `credential` is an element of type `Credential` passed to a method.

Data Flow Management Web Service

The HP Data Flow Management Web Service is an API used to integrate applications with HP BSM. The API provides methods to:

- **Manage credentials.** View, add, update, and remove.
- **Manage jobs.** View status, activate, and deactivate.
- **Manage probe ranges.** View, add, and update.
- **Manage triggers.** Add or remove a trigger CI, and add, remove, or disable a trigger TQL.
- **View general data.** Data on domains and probes.

Users of the HP Data Flow Management Web Service should be familiar with:

- The SOAP specification
- An object-oriented programming language such as C++, C# or Java
- HP BSM
- Data Flow Management

Permissions

The administrator provides login credentials for connecting with the Web service. The permission levels are View, Update, and Execute. To view the permissions required for each operation, see each operation's request documentation in the *HP Discovery and Dependency Mapping Schema Reference*.

Call the Web Service

The HP Discovery and Dependency Mapping Web Service enables calling server-side methods using standard SOAP programming techniques. If the statement cannot be parsed or if there is a problem invoking the method, the API methods throw a `SoapFault` exception. When a `SoapFault` exception is thrown, the service populates one or more of the error message, error code, and exception message fields. If there is no error, the results of the invocation are returned.

To call the service, use:

- Protocol: `http` or `https` (depending on server configuration)
- URL: `<BSM Data Processing>:21212 /axis2/services/DiscoveryService`
- Default password: `"admin"`
- Default username: `"admin"`

SOAP programmers can access the WSDL at:

- `axis2/services/DiscoveryService?wsdl`

Data Flow Management Methods

This section contains a list of the Web service operations and a brief summary of their use. For full documentation of the request and response for each operation, see the *HP Discovery and Dependency Mapping Schema Reference*.

This section includes the following topics:

- ["Data Structures" below](#)
- ["Managing Discovery Job Methods" on the next page](#)
- ["Managing Trigger Methods" on page 334](#)
- ["Domain and Probe Data Methods" on page 336](#)
- ["Credentials Data Methods" on page 338](#)
- ["Data Refresh Methods" on page 340](#)

Data Structures

These are some of the data structures used in the Data Flow Management Web Service API.

CIProperties

`CIProperties` is a collection of collections. Each collection contains properties of a different data type. For example, there can be a `dateProps` collection, a `strListProps` collection, an `xmlProps` collection, and so on.

Each type collection contains individual properties of the given type. The names of these properties elements is the same as the container, but in singular. For example, `dateProps` contains `dateProp` elements. Each property is a name-value pair.

See `CIProperties` in the *HP Discovery and Dependency Mapping Schema Reference*.

IPList

A list of `IP` elements, each of which contains an IPv4 Address.

See `IPList` in the *HP Discovery and Dependency Mapping Schema Reference*.

IPRange

An `IPRange` has two elements, the `Start` and the `End` elements. Each contains an `Address` element which is an IPv4 Address.

See `IPRange` in the *HP Discovery and Dependency Mapping Schema Reference*.

Scope

Two `IPRanges`. `Exclude` is a collection of `IPRanges` to exclude from the job. `Include` is a collection of `IPRanges` to include in the job.

See `Scope` in the *HP Discovery and Dependency Mapping Schema Reference*

Managing Discovery Job Methods

activateJob

Activates the specified job.

See "Code Sample" on page 342

Input

Parameter	Comment
cmdbContext	For details, see "CmdbContext" on page 272.
JobName	The name of the job.

deactivateJob

Deactivates the specified job.

Input

Parameter	Comment
cmdbContext	For details, see "CmdbContext" on page 272.
JobName	The name of the job.

dispatchAdHocJob

Dispatches a job on the probe ad-hoc. The job must be active and contain the specified trigger CI.

Input

Parameter	Comment
cmdbContext	For details, see "CmdbContext" on page 272.
JobName	The name of the job.
CIID	The ID of the trigger CI.
ProbeName	The name of the probe.
Timeout	In milliseconds

getDiscoveryJobsNames

Returns the list of job names.

Input

Parameter	Comment
cmdbContext	For details, see "CmdbContext" on page 272.

Output

Parameter	Comment
strList	The list of job names.

isJobActive

Checks whether the job is active.

Input

Parameter	Comment
cmdbContext	For details, see " CmdbContext " on page 272.
JobName	The name of the job to check.

Output

Parameter	Comment
JobState	True if the job is active.

Managing Trigger Methods

addTriggerCI

Adds a new trigger CI to the specified job.

Input

Parameter	Comment
cmdbContext	For details, see " CmdbContext " on page 272.
JobName	The name of the job.
CIID	The ID of the trigger CI.

addTriggerTQL

Adds a new trigger TQL to the specified job.

Input

Parameter	Comment
cmdbContext	For details, see " CmdbContext " on page 272.
JobName	The name of the job.
TqlName	The name of the TQL to add.

disableTriggerTQL

Prevents the TQL from triggering the job, but does not permanently remove it from the list of queries

that trigger the job.

Input

Parameter	Comment
cmdbContext	For details, see " CmdbContext " on page 272.
JobName	The name of the job.

removeTriggerCI

Removes the specified CI from the list of CIs that trigger the job.

Input

Parameter	Comment
cmdbContext	For details, see " CmdbContext " on page 272.
JobName	The job name.
CIID	The ID of the trigger CI.

removeTriggerTQL

Removes the specified TQL from the list of queries that trigger the job.

Input

Parameter	Comment
cmdbContext	For details, see " CmdbContext " on page 272.
JobName	Collection of job names to check.
CIID	The ID of the TQL to remove.

setTriggerTQLProbesLimit

Restrict the probes in which the TQL is active in the job to the specified list.

Input

Parameter	Comment
cmdbContext	For details, see " CmdbContext " on page 272.
JobName	The name of the job.
tqlName	The TQL name.
probesLimit	The list of probes for which the TQL is active.

Domain and Probe Data Methods

getDomainType

Returns the domain type.

Input

Parameter	Comment
cmdbContext	For details, see "CmdbContext" on page 272.
domainName	The name of the domain.

Output

Parameter	Comment
domainType	The domain type.

getDomainsNames

Returns the names of the current domains.

See ["Code Sample" on page 342](#)

Input

Parameter	Comment
cmdbContext	For details, see "CmdbContext" on page 272.

Output

Parameter	Comment
domainNames	The list of domain names.

getProbeIPs

Returns the IP addresses of the specified probe.

Input

Parameter	Comment
cmdbContext	For details, see "CmdbContext" on page 272.
domainName	The domain to check.
probeName	The name of the probe used on that domain.

Output

Parameter	Comment
probeIPs	The "IPList" of the addresses in the probe.

getProbesNames

Returns the names of the probes in the specified domain.

See ["Code Sample" on page 342](#)

Input

Parameter	Comment
cmdbContext	For details, see "CmdbContext" on page 272 .
domainName	The domain to check.

Output

Parameter	Comment
probesName	The list of probes on the domain.

getProbeScope

Returns the scope definition of the specified probe.

Input

Parameter	Comment
cmdbContext	For details, see "CmdbContext" on page 272 .
domainName	The domain to check.
probeName	The name of the probe.

Output

Parameter	Comment
probeScope	The "Scope" of the probe.

isProbeConnected

Checks whether the specified probe is connected.

See ["Code Sample" on page 342](#)

Input

Parameter	Comment
cmdbContext	For details, see " CmdbContext " on page 272.
domainName	The domain to check.
probeName	The probe to check

Output

Parameter	Comment
isConnected	True if the probe is connected.

updateProbeScope

Sets the scope of the specified probe, overriding the existing scope.

Input

Parameter	Comment
cmdbContext	For details, see " CmdbContext " on page 272.
domainName	The domain.
probeName	The probe to update.
newScope	The " Scope " to set for the probe.

Credentials Data Methods

addCredentialsEntry

Adds a credentials entry to the specified protocol for the specified domain.

See "[Code Sample](#)" on page 342

Input

Parameter	Comment
cmdbContext	For details, see " CmdbContext " on page 272.
domainName	The domain to update.
protocolName	The name of the protocol.
credentialsEntryParameters	The " CIProperties " collection of the new credentials.

Output

Parameter	Comment
credentialsEntryID	The CI ID of the new credential entry.

getCredentialsEntriesIDs

Returns the IDs of the credentials defined for the specified protocol.

Input

Parameter	Comment
cmdbContext	For details, see " CmdbContext " on page 272.
domainName	The domain to get the credentials for.
protocolName	The name of a protocol used on that domain.

Output

Parameter	Comment
credentialsEntryIDs	The list of credential IDs for the protocol on the domain.

getCredentialsEntry

Returns the credentials defined for the specified protocol. Encrypted attributes are returned empty.

Input

Parameter	Comment
cmdbContext	For details, see " CmdbContext " on page 272.
domainName	The domain to get the credentials for.
protocolName	A protocol used on that domain.
credentialsEntryID	The credential ID to get.

Output

Parameter	Comment
credentialsEntryParameters	The " CIProperties " collection of the credentials.

removeCredentialsEntry

Removes the specified credentials from the protocol.

Input

Parameter	Comment
cmdbContext	For details, see " CmdbContext " on page 272.
domainName	The domain.
protocolName	A protocol used on the domain.
credentialsEntryID	The ID of the credential to remove.

updateCredentialsEntry

Sets new values for properties of the specified credentials entry.

The existing properties are deleted and these properties are set. Any property whose value is not set in this call is left undefined.

Input

Parameter	Comment
cmdbContext	For details, see " CmdbContext " on page 272.
domainName	The domain to update credentials in.
protocolName	A protocol used on the domain.
credentialsEntryID	The ID of the credentials to update.
credentialsEntryParameters	The " CIProperties " collection to set as properties for the credentials.

Data Refresh Methods

rediscoverCIs

Locates the triggers that discovered the specified CI objects and reruns those triggers.

rediscoverCIs runs asynchronously. Call **checkDiscoveryProgress** to determine when the rediscovery is complete.

Input

Parameter	Comment
cmdbContext	For details, see " CmdbContext " on page 272.
CmdbIDs	Collection of IDs of the objects to rediscover.

Output

Parameter	Comment
isSucceed	True if the CIs rediscovery succeeded.

checkDiscoveryProgress

Returns the progress of the most recent **rediscoverCIs** call on the specified IDs. The response is a value from 0 to 1. When the response is 1, the **rediscoverCIs** call has completed.

Input

Parameter	Comment
cmdbContext	For details, see " CmdbContext " on page 272.
CmdbIDs	Collection of IDs of the objects in the rediscover call to track.

Output

Parameter	Comment
progress	A completed job has a progress of 1. Jobs that have not completed have a fraction less than 1.

rediscoverViewCIs

Locates the triggers that created the data to populate the specified view, and reruns those triggers.

rediscoverViewCIs runs asynchronously. Call **checkViewDiscoveryProgress** to determine when the rediscovery is complete.

Input

Parameter	Comment
cmdbContext	For details, see " CmdbContext " on page 272.
viewName	The views to check.

Output

Parameter	Comment
isSucceed	True if CIs rediscovery succeeded.

checkViewDiscoveryProgress

Returns the progress of the most recent **rediscoverViewCIs** call on the specified view. The response is a value from 0 to 1. When the response is 1, the **rediscoverCIs** call has completed.

Input

Parameter	Comment
cmdbContext	For details, see " CmdbContext " on page 272.
viewName	The collection of views to check.

Output

Parameter	Comment
progress	A completed job has a progress of 1. Jobs that have not completed have a fraction less than 1.

Code Sample

```
import java.net.URL;
import org.apache.axis2.transport.http.HTTPConstants;
import org.apache.axis2.transport.http.HttpTransportProperties;
import com.hp.ucmdb.generated.params.discovery.*;
import com.hp.ucmdb.generated.services.*;
import com.hp.ucmdb.generated.types.*;
public class test {
    static final String HOST_NAME = "<my_hostname>";
    static final int PORT = 21212;
    private static final String PROTOCOL = "http";
    private static final String FILE =
"/axis2/services/DiscoveryService";

    private static final String PASSWORD = "<my_password>";
    private static final String USERNAME = "<my_username>";

    private static CmdbContext cmdbContext = new CmdbContext("ws
tests");

    public static void main(String[] args) throws Exception {
        // Get the stub object
        DiscoveryService discoveryService = getDiscoveryService();

        // Activate Job
        discoveryService.activateJob(new ActivateJobRequest(
            "Range IPs by ICMP", cmdbContext));

        // Get domain & probes info
        getProbesInfo(discoveryService);
        // Add credentilas entry for ntcmd protocol
        addNTCMDCredentialsEntry();
    }

    public static void addNTCMDCredentialsEntry() throws Exception {
        DiscoveryService discoveryService = getDiscoveryService();

        // Get domain name
        StrList domains =
            discoveryService.getDomainsNames(
                new GetDomainsNamesRequest(cmdbContext)).
                getDomainNames();
        if (domains.sizeStrValueList() == 0) {
            System.out.println("No domains were found, can't create
credentials");
            return;
        }
    }
}
```

```
String domainName = domains.getStrValue(0);
// Create properties with one byte param
CIProperties newCredsProperties = new CIProperties();

// Add password property - this is of type bytes
newCredsProperties.setBytesProps(new BytesProps());
setPasswordProperty(newCredsProperties);

// Add user & domain properties - these are of type string
newCredsProperties.setStrProps(new StrProps());
setStringProperties("protocol_username", "test user",
newCredsProperties);
setStringProperties("ntadminprotocol_ntdomain",
    "test doamin", newCredsProperties);

// Add new credentials entry
discoveryService.addCredentialsEntry(
    new AddCredentialsEntryRequest(domainName,
        "ntadminprotocol", newCredsProperties, cmdbContext));
System.out.println("new credentials created for domain: " +
domainName + " in ntcmd protocol");
}

private static void setPasswordProperty(CIProperties
newCredsProperties) {
    BytesProp bProp = new BytesProp();
    bProp.setName("protocol_password");
    bProp.setValue(new byte[] {101,103,102,104});
    newCredsProperties.getBytesProps().addBytesProp(bProp);
}

private static void setStringProperties(String propertyName,
String value, CIProperties newCredsProperties) {
    StrProp strProp = new StrProp();
    strProp.setName(propertyName);
    strProp.setValue(value);
    newCredsProperties.getStrProps().addStrProp(strProp);
}

private static void getProbesInfo(DiscoveryService
discoveryService) throws Exception {
    GetDomainsNamesResponse result =
discoveryService.getDomainsNames(new GetDomainsNamesRequest
(cmdbContext));
    // Go over all the domains
    if (result.getDomainNames().sizeStrValueList() > 0) {
        String domainName =
            result.getDomainNames().getStrValue(0);
```

```
        GetProbesNamesResponse probesResult =
            discoveryService.getProbesNames(
                new GetProbesNamesRequest(domainName,
cmdbContext));
        // Go over all the probes
        for (int i=0; i<probesResult.getProbesNames
().sizeStrValueList(); i++) {
            String probeName = probesResult.getProbesNames
().getStrValue(i);
            // Check if connected
            IsProbeConnectedResponse connectedRequest =
                discoveryService.isProbeConnected(
                    new IsProbeConnectedRequest(
                        domainName, probeName, cmdbContext));
            Boolean isConnected = connectedRequest.getIsConnected
();
            // Do something ...
            System.out.println("probe " + probeName + "
isconnect=" + isConnected);
        }
    }

    private static DiscoveryService getDiscoveryService() throws
Exception {
        DiscoveryService discoveryService = null;
        try {
            // Create service
            URL url = new URL(PROTOCOL,HOST_NAME,PORT, FILE);
            DiscoveryServiceStub serviceStub =
                new DiscoveryServiceStub(url.toString());

            // Authenticate info
            HttpTransportProperties.Authenticator auth =
                new HttpTransportProperties.Authenticator();
            auth.setUsername(USERNAME);
            auth.setPassword(PASSWORD);
            serviceStub._getServiceClient().getOptions().setProperty(
                HTTPConstants.AUTHENTICATE, auth);

            discoveryService = serviceStub;
        } catch (Exception e) {
            throw new Exception("cannot create a connection to service
", e);
        }
        return discoveryService;
    }
}
```