

HP Universal CMDB

for the Windows and Linux operating systems

Software Version: 9.00

Developer Reference

Document Release Date: June 2010

Software Release Date: June 2010



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 Notices

© Copyright 2005 - 2010 Hewlett-Packard Development Company, L.P.

Trademark Notices

Adobe® and Acrobat® are trademarks of Adobe Systems Incorporated.

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 US trademark of Sun Microsystems, Inc.

Microsoft®, Windows®, Windows NT®, Windows® XP, and Windows Vista® are U.S. registered trademarks of Microsoft Corporation.

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 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/officeInfs>).
- This product includes JEP - Java Expression Parser code from Netaphor Software, Inc. (<http://www.netaphor.com/home.asp>).

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.

Support

Visit the HP Software Support 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

Table of Contents

Welcome to This Guide	11
How This Guide Is Organized	11
Who Should Read This Guide	12
HP Universal CMDB Online Documentation	12
Additional Online Resources.....	13
Documentation Updates	14

PART I: CREATING DISCOVERY AND INTEGRATION ADAPTERS

Chapter 1: Adapter Development and Writing	17
Adapter Development and Writing Overview	18
Associating Business Value with Discovery Development	19
Adapters and Related Components	20
The Adapter Development Cycle	21
Data Flow Management and Integration	24
Research Stage	26
Separating Adapters.....	30
Using External Java JAR Files Within Jython.....	31
HP Data Flow Management API Reference	32
Implement an Adapter	32
Step 1: Create an Adapter.....	33
Step 2: Assign a Job to the Adapter	43
Step 3: Create Jython Code	45
Record DFM Code.....	59
Data Flow Management Code.....	61
Jython Libraries and Utilities	64
Chapter 2: Configuring Eclipse to Run Jython Scripts.....	71
Configure Eclipse to Run Jython Scripts in Debug Mode.....	71
Chapter 3: Debugging Content With Discovery Analyzer	81
Using Discovery Analyzer to Debug Content Overview.....	81
Work with Discovery Analyzer	83
Troubleshooting and Limitations	89

Chapter 4: Supporting Multi-Lingual Locales	91
Supporting Multi-Lingual Locales Overview	91
Determining the Character Set for Encoding	92
Resource Bundles.....	94
Add Support for a New Language	95
Define a New Job to Operate With Localized Data.....	97
Decode Commands Without a Keyword	98
Change the Default Language.....	99
Write Error Messages for a Specific Locale	99
API Reference.....	100
Chapter 5: Error Messages	105
Error Messages Overview	106
Write Error Messages	107
Error-Writing Conventions	107
Error Severity Levels	110
Troubleshooting and Limitations	111
Chapter 6: Integration Framework SDK	113
Federation Framework – Overview.....	114
Adapter and Mapping Interaction with the Federation Framework	118
Federation Framework Flow for FTQL.....	119
Federation Framework Flow for Replication	131
Adapter Interfaces.....	133
Add an Adapter for a New External Data Source	134
Implement the Default Mapping Engine.....	141
Add a New Adapter – Scenario	142
Adapter Capabilities	148

Chapter 7: Generic Database Adapter	151
Generic Database Adapter Overview.....	153
Non-supported TQL Queries	153
Reconciliation.....	154
Hibernate as JPA Provider.....	155
Validate Adapter Setup.....	157
Prepare the Adapter Package	162
Configure the Adapter.....	164
Deploy the Adapter	172
Edit the Adapter.....	173
Load the Adapter	173
Create an Integration Point.....	174
Create a View.....	174
Calculate the Results	176
View the Results	177
View Reports.....	178
Enable Log Files	179
Federated Database Configuration Files.....	179
The adapter.conf File.....	180
The simplifiedConfiguration.xml File.....	181
The orm.xml File	183
The reconciliation_types.txt file	189
The reconciliation_rules.txt File (for backwards compatibility)	189
The transformations.txt File.....	192
The persistence.xml File	193
The discriminator.properties File	194
The replication_config.txt File	196
The fixed_values.txt File.....	196
Out-of-the-Box Converters.....	196
Plug-ins.....	200
Configuration Examples.....	201
Federated Database Log Files.....	211
External References	214
Troubleshooting and Limitations	214
Chapter 8: HP ServiceCenter/Service Manager Adapter	217
Adapter Usage.....	218
The Adapter Configuration File	220
Multi-Threading	228
Deploy the Adapter – Typical Deployment.....	228
Deploy the ServiceDesk Adapter	229
Add an Attribute to the ServiceCenter/Service Manager CIT	233
Communicate with Service Manager over SSL	241

Chapter 9: Mapping Between CIT Attributes and Database Tables 243
 Use Eclipse to Map Between CIT Attributes and Database Tables243

PART II: APIS

Chapter 10: Introduction to APIs265
 APIs Overview.....265

Chapter 11: HP Universal CMDB Web Service API267
 Conventions268
 Using the HP Universal CMDB Web Service API268
 HP Universal CMDB Web Service API Reference270
 Returning Unambiguous Topology Map Elements.....270
 Call the Web Service274
 Query the UCMDB274
 Update the UCMDB279
 Query the UCMDB Class Model281
 Query for Impact Analysis.....283
 UCMDB Query Methods283
 UCMDB Update Methods298
 UCMDB Impact Analysis Methods301
 Use Cases304
 Examples.....305
 UCMDB General Parameters336
 UCMDB Output Parameters340

Chapter 12: HP Universal CMDB API343
 Conventions343
 Using the HP Universal CMDB API.....344
 General Structure of an Application345
 Put the API Jar File in the Classpath346
 Create an Integration User346
 HP Universal CMDB API Reference.....347
 Use Cases347
 Examples.....349

Chapter 13: Data Flow Management Web Service API355
 Data Flow Management Web Service API Overview355
 Data Flow Management Web Service API356
 Data Flow Management Methods357
 Data Flow Management Adding Credentials Example360

Index367

Welcome to This Guide

This guide explains how to create and manage adapters that enable you to send and receive data from external data repositories and other CMDBs.

This chapter includes:

- ▶ How This Guide Is Organized on page 11
- ▶ Who Should Read This Guide on page 12
- ▶ HP Universal CMDB Online Documentation on page 12
- ▶ Additional Online Resources on page 13
- ▶ Documentation Updates on page 14

How This Guide Is Organized

The guide contains the following chapters:

Part I Creating Discovery and Integration Adapters

Describes how to create adapters.

Part II APIs

Describes how to work with the APIs to extract configuration data from HP Universal CMDB.

Who Should Read This Guide

This guide is intended for the following users of HP Universal CMDB:

- HP Universal CMDB administrators
- HP Universal CMDB platform administrators
- HP Universal CMDB application administrators
- HP Universal CMDB data management administrators

Readers of this guide should be knowledgeable about enterprise system administration, have familiarity with ITIL concepts, and be knowledgeable about HP Universal CMDB.

HP Universal CMDB Online Documentation

HP Universal CMDB includes the following online documentation:

Readme. Provides a list of version limitations and last-minute updates. From the HP Universal CMDB DVD root directory, double-click **readme.html**. You can also access the most updated readme file from the HP Software Support Web site.

What's New. Provides a list of new features and version highlights. In HP Universal CMDB, select **Help > What's New**.

Printer-Friendly Documentation. Choose **Help > UCMDB Help**. The following guides are published in PDF format only:

- **Deployment.** Explains the hardware and software requirements needed to set up HP Universal CMDB, how to install or upgrade HP Universal CMDB, how to harden the system, and how to log in to the application.
- **Database.** Explains how to set up the database (MS SQL Server or Oracle) needed by HP Universal CMDB.
- **Discovery and Integrations Content.** Explains how to run discovery to discover applications, operating systems, and network components running on your system. Also explains how to discover data on other data repositories through integration.

HP Universal CMDB Online Help includes:

- ▶ **Modeling.** Enables you to manage the content of your IT Universe model.
- ▶ **Data Flow Management.** Explains how to integrate HP Universal CMDB with other data repositories and how to set up HP Universal CMDB to discover network components.
- ▶ **Administration.** Explains how to work with HP Universal CMDB.
- ▶ **Developer Reference.** For users with an advanced knowledge of HP Universal CMDB. Explains how to define and use adapters and how to use APIs to access data.

Online Help is also available from specific HP Universal CMDB windows by clicking in the window and clicking the **Help** button.

Online books can be viewed and printed using Adobe Reader, which can be downloaded from the Adobe Web site (www.adobe.com).

Additional Online Resources

Troubleshooting & Knowledge Base. Enables you to search the Self-solve knowledge base in the Troubleshooting page on the HP Software Support Web site. Choose **Help > Troubleshooting & Knowledge Base**. The URL for this Web site is <http://h20230.www2.hp.com/troubleshooting.jsp>.

HP Software Support. Enables you to browse the Self-solve knowledge base. You can also post to and search user discussion forums, submit support requests, download patches and updated documentation, and more. Choose **Help > HP Software Support**. The URL for this Web site is www.hp.com/go/hpssoftwaresupport.

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract.

To find more information about access levels, go to:

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

To register for an HP Passport user ID, go to:

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

HP Software Web site. Provides you with the most up-to-date information on HP Software products. This includes new software releases, seminars and trade shows, customer support, and more. Choose **Help > HP Software Web site.** The URL for this Web site is www.hp.com/go/software.

Documentation Updates

HP Software is continually updating its product documentation with new information.

To check for recent updates, or to verify that you are using the most recent edition of a document, go to the HP Software Product Manuals Web site (<http://h20230.www2.hp.com/selfsolve/manuals>).

Part I

Creating Discovery and Integration Adapters

1

Adapter Development and Writing

This chapter includes:

Concepts

- ▶ Adapter Development and Writing Overview on page 18
- ▶ Associating Business Value with Discovery Development on page 19
- ▶ Adapters and Related Components on page 20
- ▶ The Adapter Development Cycle on page 21
- ▶ Data Flow Management and Integration on page 24
- ▶ Research Stage on page 26
- ▶ Separating Adapters on page 30
- ▶ Using External Java JAR Files Within Jython on page 31
- ▶ HP Data Flow Management API Reference on page 32

Tasks

- ▶ Implement an Adapter on page 32
- ▶ Step 1: Create an Adapter on page 33
- ▶ Step 2: Assign a Job to the Adapter on page 43
- ▶ Step 3: Create Jython Code on page 45
- ▶ Record DFM Code on page 59

Reference

- ▶ Data Flow Management Code on page 61
- ▶ Jython Libraries and Utilities on page 64

Concepts

Adapter Development and Writing Overview

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

The following sections can help you understand what you need to know and do, to successfully manage and execute a discovery development project.

This chapter:

- ▶ Assumes a working knowledge of HP Universal CMDB 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 HP Universal CMDB, 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 Universal Data Flow Management 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.)

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 CMDB 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 CMDB, 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 CMDB to provide business value.

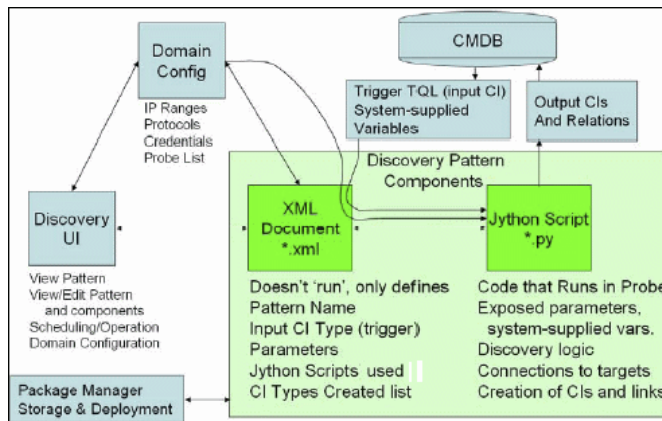
One method could be to provide a questionnaire to the application owner (also the SME in this case). The owner should be able to specify the above goals and blueprint. The owner must at least provide a current architecture of the application.

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.

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 CMDDB 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.

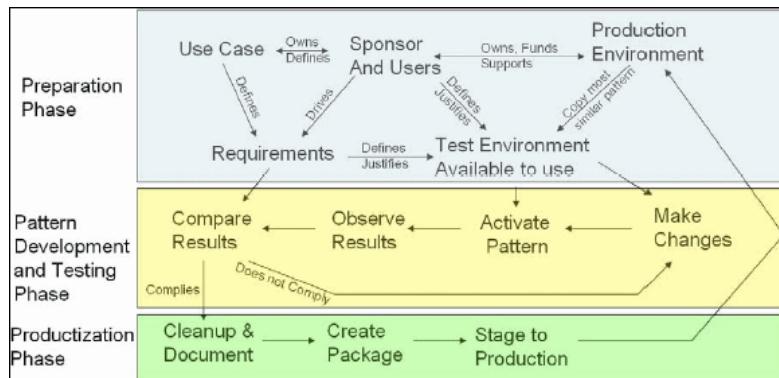
Input CIs to the adapter are provided by a TQL, and are exposed to the adapter script in system-supplied variables. Adapter parameters are also supplied as destination data, so you can configure the adapter's operation according to an adapter's specific function.

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.

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.



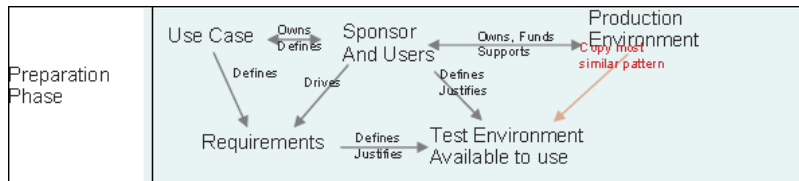
Each phase of adapter development builds on the last one.

Once you are satisfied with the way the adapter looks and works, you are ready to package it. Using either the UCMDB 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 UCMDB 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 *HP UCMDB Administration Guide*.

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

- ▶ Research and Preparation Phase
- ▶ Adapter Development and Testing
- ▶ Adapter Packaging and Productization

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 *HP Universal CMDB 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

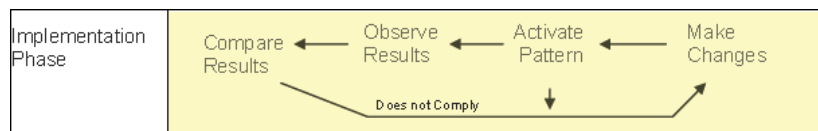
- You now know what a similar adapter should look like

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 *HP UCMDB 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 *HP UCMDB 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, UCMDB CIs for input.

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

From HP Universal CMDB 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 HP Universal CMDB: 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

Research Stage

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 18.

This section includes the following topics:

- "Modifying an Existing Adapter" on page 26
- "Writing a New Adapter" on page 27
- "Model Research" on page 27
- "Technology Research" on page 28
- "Guidelines for Choosing Ways to Access Data" on page 28
- "Summary" on page 30

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 CMDB (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 CMDB 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 HP Universal CMDB.

Technology Research

Once you have verified that the CMDB hold 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 HP Universal CMDB 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 HP Universal CMDB connection protocols: **Discovery > Setup Discovery Probe > Domains and Probes pane**. For details, see "Domains and Probes Pane" in the *HP Universal CMDB 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 CMDB 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 issues 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 UCMDDB model).

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.

Using 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 HP Universal CMDB 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.

HP Data Flow Management API Reference

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

C:\hp\UCMDB\UCMDBServer\j2f\AppServer\webapps\site.war\amdocs\eng\doc_lib\DevRef_guide\DDM_JavaDoc\index.html

Tasks

Implement an Adapter

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

1 DFM 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 the following section.

2 Discovery job.

You configure a job with scheduling information and a trigger TQL. For details, see "Step 2: Assign a Job to the Adapter" on page 43.

3 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 45.

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.

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 *HP Universal CMDB 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 "Trigger CIs and Trigger Queries" in the *HP Universal CMDB Data Flow Management Guide*.

This section includes the following topics:

- "Define Adapter Input (Trigger CIT and Input Query)" on page 34
- "Define Adapter Output" on page 40
- "Override Adapter Parameters" on page 41

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 CMDB. 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 page 36 and "Add Query Nodes and Relationships to a TQL Query" in *Modeling Guide*.

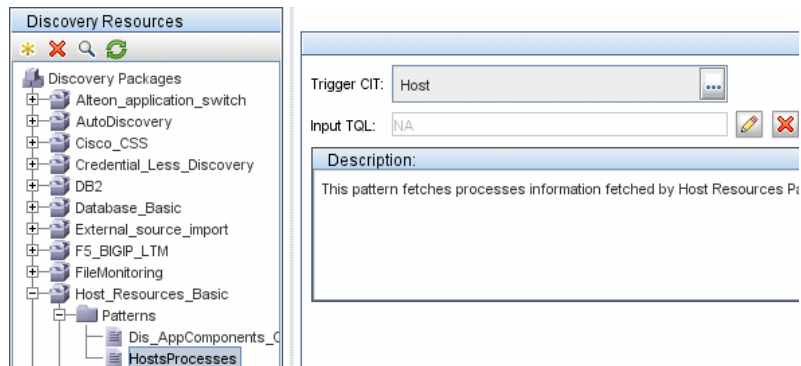
- ▶ The Trigger CI data contains all the required information on the Trigger CI as well as information from the other nodes in the Input TQL, if they are defined. DFM uses variables to retrieve data from the CIs. When the task is downloaded to the Probe, the Trigger CI data variables are replaced with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

- 1** Access **Discovery > Manage Discovery Resources > Adapter Signature**. Select the HostProcesses adapter (**Discovery Packages > Host_Resources_Basic > Patterns > HostProcesses**).
- 2** Locate the Trigger CIT box. For details, see "Triggered CI Data Pane" in the *HP Universal CMDB Data Flow Management Guide*. Click the button to open the Choose Discovered Class dialog box. For details, see "Choose Discovered Class Dialog Box" in the *HP Universal CMDB Data Flow Management Guide*.
- 3** Select the CIT.

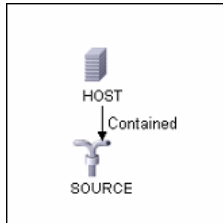
In this example, the IP CI (Host) is permitted in the adapter:



Example of Input Query Definition

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

- 1** Access **Discovery > Manage Discovery Resources > Pattern Signature**. Locate the Input TQL box. Click the **Edit** button to open the Input TQL Editor. For details, see "Input Query Editor Window" in the *HP Universal CMDB Data Flow Management Guide*.
- 2** In the Input TQL Editor, name the Trigger CI node **SOURCE**: right-click the node and choose **Node Properties**. In the **Element Name** box, change the name to **SOURCE**.
- 3** Add a Host CI and a **Contains** relationship to the IP CI. For details on working with the Input TQL Editor, see "Input Query Editor Window" in the *HP Universal CMDB Data Flow Management Guide*.

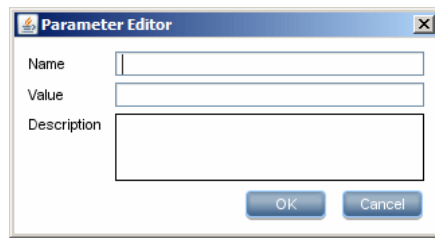


The **IP CI** is connected to a **HOST CI**. The input TQL consists of two nodes, **HOST** and **IP**, with a link between them. The **IP CI** is named **SOURCE**.

Example of Adding Variables to the Input TQL:

In this example, you add DIRECTORY and CONFIGURATION_FILE variables to the Input TQL 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.

- 1 Display the Input TQL created in the previous example.
- 2 Access **Discovery > Manage Discovery Resources > Pattern Signature**. Locate the Triggered CI Data pane. For details, see "Triggered CI Data Pane" in the *HP Universal CMDB Data Flow Management Guide*.
- 3 Add variables to the Input TQL. For details, see the Value field in the "Triggered CI Data Pane" in the *HP Universal CMDB Data Flow Management Guide*.



Example of Replacing Variables with Actual Data:

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

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

Triggered CI Data	
Name	Value
Protocol	\${SOURCE.credentials_id}
credentialsId	\${SOURCE.credentials_id}
fileName	\${CONFIGURATION_FILE.data_name}
hostID	\${HOST.root_id}
ip_address	\${SOURCE.application_ip}
path	\${CONFIGURATION_FILE.document_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 File node (defined in the Input TQL – see previous example).

Input TQL Editor

TQL Name :

Mode: Layout:

SOURCE

Configuration File

Container link

Element Name: Configuration File
CI Type: Configuration File
Visible: true
Condition: Document Path Equal c:\temp
AND Name Equal data_name
Cardinality: Container link (SOURCE, Configuration File) : 1..*

CI Type Selector

- IT Universe (434507)
 - Business (11)
 - IT Process (0)
 - Monitor (0)
 - System (434496)
 - Application Resource (233)
 - Database Resource (1)
 - Document (2233)
 - Configuration File**
 - DomainController Reso
 - HTTP Context (0)
 - IBM MQ Channel (0)
 - Q Cluster (0)
 - Q Queue (0)
 - Q Queue Mana
 - source (5949)
 - Managed Object
 - JDBC Data Source (90)
 - License Resource (0)
 - Microsoft Exchange R
 - Oracle AS Resource (
 - Oracle E-Business Su
 - Resource Pool (0)
 - SAP Resource (4095)
 - Siebel Application (0)
 - Siebel Component (0)
 - Siebel Component Gro
 - Siebel Web Application
 - Siebel Web Server Ext
 - Web Server Virtual Ho
 - WebService Resource
 - Application System (485)
 - Host (5821)
 - Host Resource (57289)

Information Pane

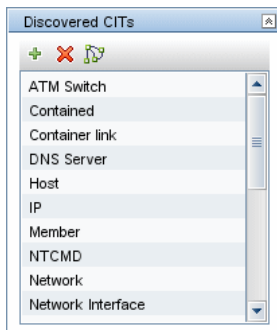
Document Path Equal c:\temp
AND Name Equal data_name

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

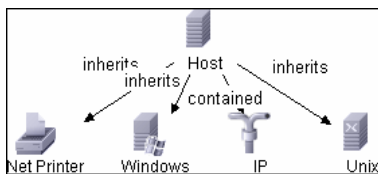
Key	Name	Display Name	Type	Description	Default Value	Visible
	ack_cleared_time	ack_cleared_time	long			
	ack_id	ack_id	string			
	BODY_ICON	BODY_ICON	string		host	
	city	City	string	City location		✓
	codepage	CodePage	string	System su...		
	contextmenu	Context Menu	string_list	Context me...	itCIs	
	country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

Define Adapter Output

The output of the adapter is a list of discovered CIs (**Discovery > Manage Discovery Resources > Pattern Signature tab**) and the links between them:



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



The discovered CIs are returned by the DFM code (that is, the Jython script) in the format of UCMDB's `ObjectStateHolderVector`. For details, see "Results Generation by the Jython Script" on page 51.

Example of Adapter Output:

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

- 1** Access **Discovery > Manage Discovery Resources**.
- 2** In the Discovery Resources pane, select **Network > Pattern > NSLOOKUP_on_Probe**.
- 3** In the Pattern Signature tab, locate the Discovered CITs pane.
- 4** The CITs that are to be part of the adapter output are listed. Add CITs to, or remove from, the list. For details, see "Discovered CITs Pane" in the *HP Universal CMDB Data Flow Management Guide*.

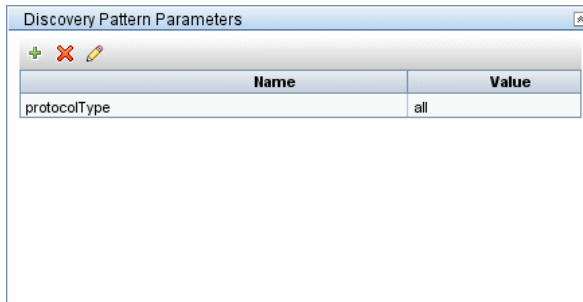
Override Adapter Parameters

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.

- 1 Access **Discovery > Manage Discovery Resources**.
- 2 In the Discovery Resources pane, select **Database Basic > Pattern > SQL_NET_Dis_Connection**.
- 3 In the Pattern Signature tab, locate the **Discovery Pattern Parameters** pane. The protocolType parameter has a value of **all**:



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

Parameters		
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.

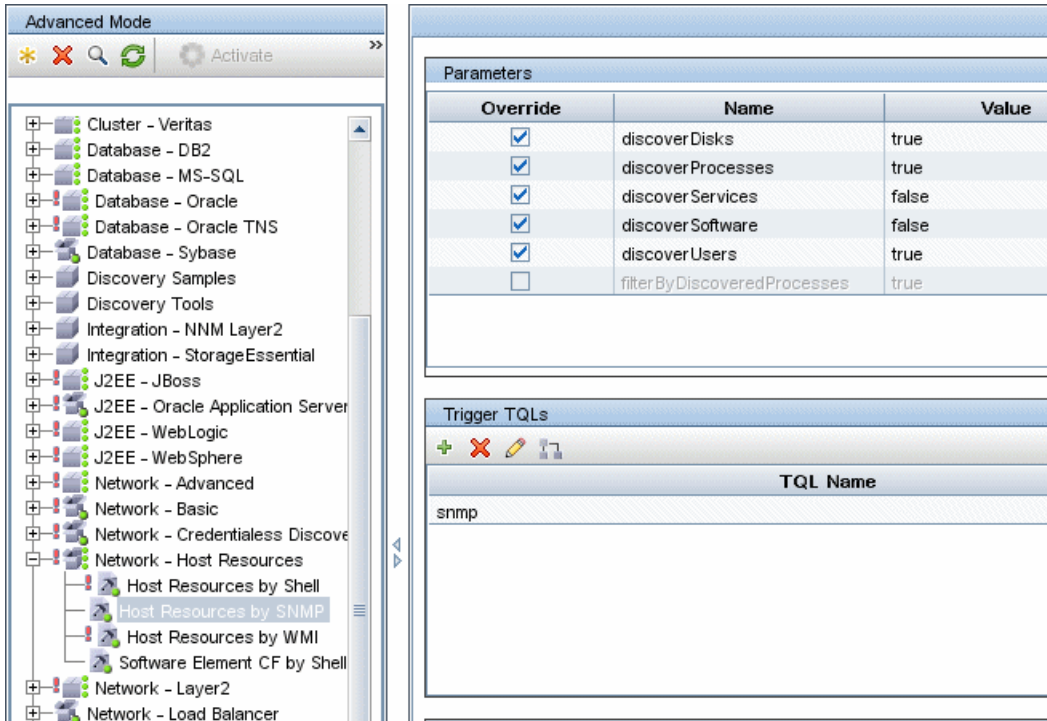
For details on adding, deleting, or editing parameters, see "Adapter Parameters Pane" in the *HP Universal CMDB Data Flow Management Guide*.

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

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 set 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.



The screenshot shows the Discovery Modules tree on the left and the configuration panel on the right. The tree includes various modules such as Cluster - Veritas, Database - DB2, Database - MS-SQL, Database - Oracle, Database - Oracle TNS, Database - Sybase, Discovery Samples, Discovery Tools, Integration - NNM Layer2, Integration - StorageEssential, J2EE - JBoss, J2EE - Oracle Application Server, J2EE - WebLogic, J2EE - WebSphere, Network - Advanced, Network - Basic, Network - Credentialless Discover, Network - Host Resources, Network - Layer2, and Network - Load Balancer. The 'Host Resources by SNMP' module is selected. The configuration panel on the right shows the 'Parameters' table and the 'Trigger TQLs' section.

Override	Name	Value
<input checked="" type="checkbox"/>	discoverDisks	true
<input checked="" type="checkbox"/>	discoverProcesses	true
<input checked="" type="checkbox"/>	discoverServices	false
<input checked="" type="checkbox"/>	discoverSoftware	false
<input checked="" type="checkbox"/>	discoverUsers	true
<input type="checkbox"/>	filterByDiscoveredProcesses	true

The 'Trigger TQLs' section shows a table with the following content:

TQL Name
snmp

Trigger TQL

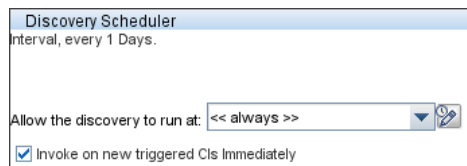
Each job is associated with Trigger TQLs. These Trigger TQLs publish results that are used as Input Trigger CIs for the adapter of this job.

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.

Scheduling

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.



Discovery Scheduler
Interval, every 1 Days.

Allow the discovery to run at: << always >>

Invoke on new triggered CIs Immediately

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 *HP Universal CMDB Data Flow Management Guide*.

Parameters

When configuring a job you can override the adapter parameters. For details, see "Override Adapter Parameters" on page 41.

Step 3: Create Jython Code

HP Universal CMDB 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 Data Flow Management API Reference*.
-

This section includes the following topics:

- "Execution of the Code" on page 46
- "Modifying Out of the Box Scripts" on page 46
- "Structure of the Jython File" on page 48
- "Results Generation by the Jython Script" on page 51
- "The Framework Instance" on page 53
- "Finding the Correct Credentials (for Connection Adapters)" on page 56
- "Handling Exceptions from Java" on page 59
- "Jython Libraries and Utilities" on page 64
- "Using External Java JAR Files Within Jython" on page 31

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 HP Universal CMDB 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 *HP Universal CMDB 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 *HP Universal CMDB Data Flow Management 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
- 2 Main Function - DiscoveryMain
- 3 Functions definitions (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:

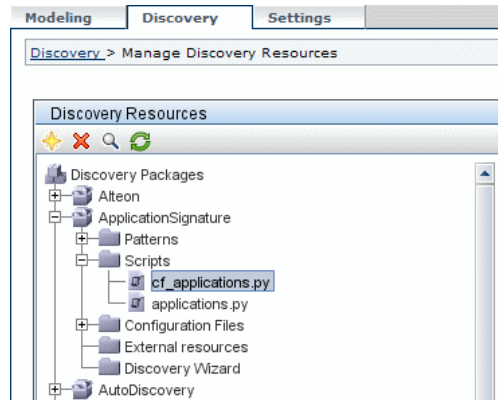
```
def DiscoveryMain(Framework):
```

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 (**Discovery > Manage Discovery Resources > Discovery Packages > AutoDiscovery > 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 CMDB. 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 CMDB. 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 CMDB 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', sblsvrName)
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 **sblsvrName** 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 CMDB.
- 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 *HP Universal CMDB 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 *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 CMDB server, see the `sendObjects` method.

Once the result graph is assembled in an `ObjectStateHolderVector` instance, it must be returned to the DFM Framework to be inserted into the CMDB. 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 64.

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 32.

This section describes the most important Framework usages:

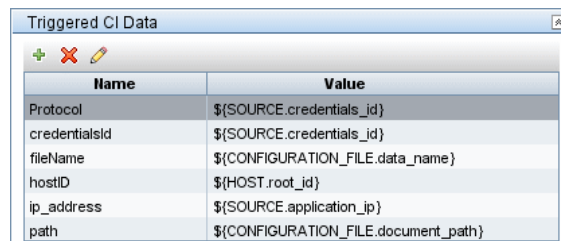
- "Framework.getTriggerCIData(String attributeName)" on page 53
- "Framework.createClient(credentialsId, props)" on page 54
- "Framework.getParameter (String parameterName)" on page 55
- "Framework.reportError(String message) and Framework.reportWarning(String message)" on page 56

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:



Name	Value
Protocol	\${SOURCE.credentials_id}
credentialsId	\${SOURCE.credentials_id}
fileName	\${CONFIGURATION_FILE.data_name}
hostID	\${HOST.root_id}
ip_address	\${SOURCE.application_ip}
path	\${CONFIGURATION_FILE.document_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. For details on credentials and supported protocols, see "Domain Credential References" in the *HP Universal CMDB 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:

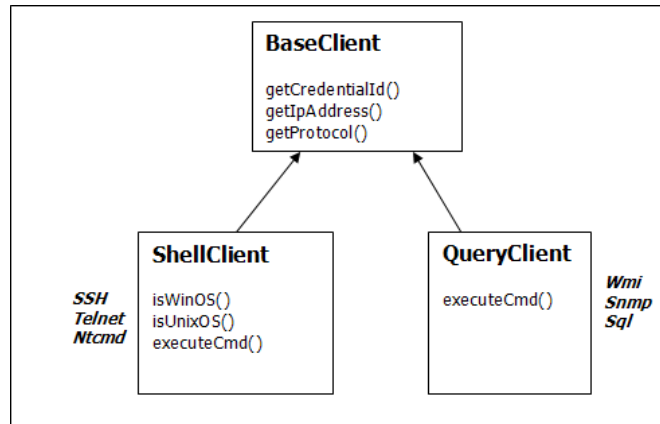
To create a WMI client and run a WMI query using the client:

```
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 Universal CMDB Data Flow Management API Reference*.

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 CMDB. 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 30.

```
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.

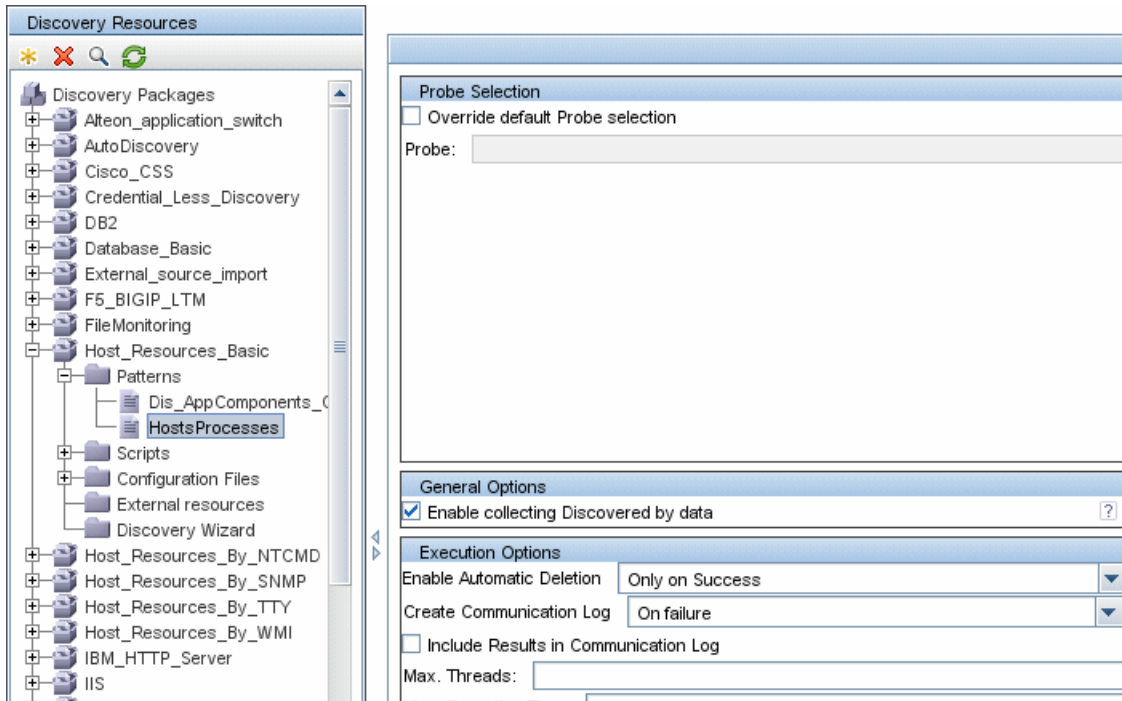
Record DFM Code

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 Discovery code:

- 1 Access **Discovery > Discovery Control Panel**. Right-click the job whose run must be logged and select **Edit adapter** to open the Manage Discovery Resources application.

2 Locate the **Execution Options** pane in the Pattern Management tab:



3 Change the **Create communication logs** box to **Always**. For details on setting logging options, see "Execution Options Pane" in the *HP Universal CMDB 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
    <frame class="com.hp.ucmdb.discovery.probe.clients.shell.SSHClient" method="createWrapper" file="SSHClient.java"
    <frame class="com.hp.ucmdb.discovery.probe.clients.BaseClient" method="initPrivate" file="BaseClient.java" />
  </stacktrace>
</error>
</exec>
```

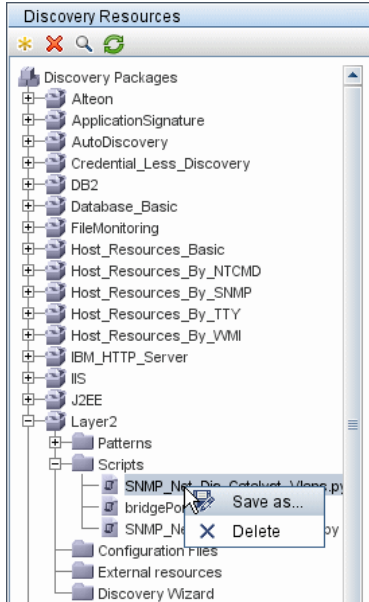
Reference

Data Flow Management Code

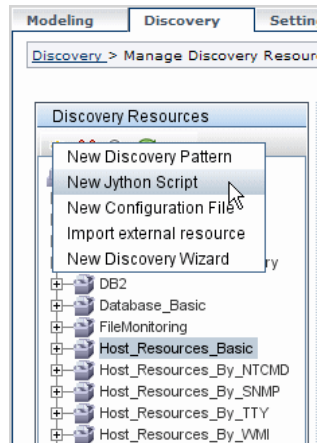
The actual implementation of connecting to the remote system, querying its data, and mapping it as CMDB data is performed by the Jython code. For example, the code contains the logic for connecting to a database and extracting data from it. In this case, the code expects to receive a JDBC URL, a user 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.

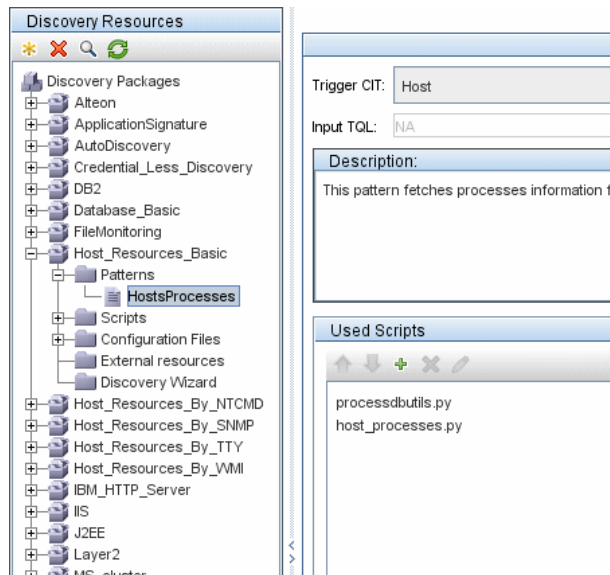
An adapter can contain a list of scripts to be used when running discovery. When creating a new adapter, you usually create a new script and assign it to the adapter. A new script includes basic templates, but you can use one of the other scripts as a template by right-clicking it and selecting **Save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



For details on writing new Jython scripts, see "Step 3: Create Jython Code" on page 45. You add scripts through the Manage Discovery Resources window:



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.

Jython Libraries and Utilities

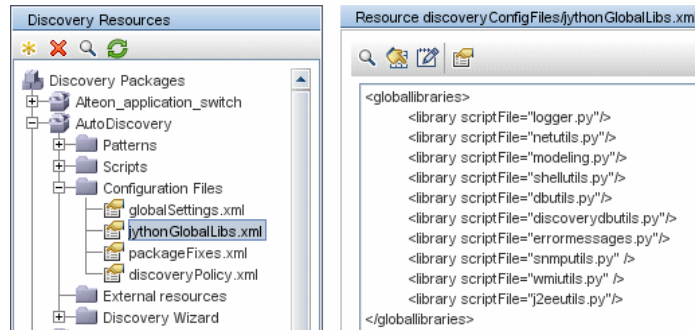
Several utility scripts are used widely in adapters. These scripts are part of the AutoDiscovery package and are located under: **C:\hp\DDM\DiscoveryProbe\root\lib\collectors\probeManager\discoveryScripts** with the other scripts that are downloaded to the Probe.

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" on page 65
- "modeling.py" on page 67
- "netutils.py" on page 67
- "shellutils.py" on page 67

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\DDM\DiscoveryProbe\root\logs\probeMgr-patternsDebug.log.

Messages are entered in the log file according to the debug level defined for the PATTERNS_DEBUG appender in the **C:\hp\DDM\DiscoveryProbe\root\lib\collectors\probeManager\probeMgrLog4j.properties** file. (By default, the level is DEBUG.) For details, see "Error Severity Levels" on page 110.

```
#####
##### 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/DDM/DiscoveryProbe/root/logs/
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..')
```


2

Configuring Eclipse to Run Jython Scripts

This chapter includes:

Tasks

- ▶ Configure Eclipse to Run Jython Scripts in Debug Mode on page 71

Tasks

Configure Eclipse to Run Jython Scripts in Debug Mode

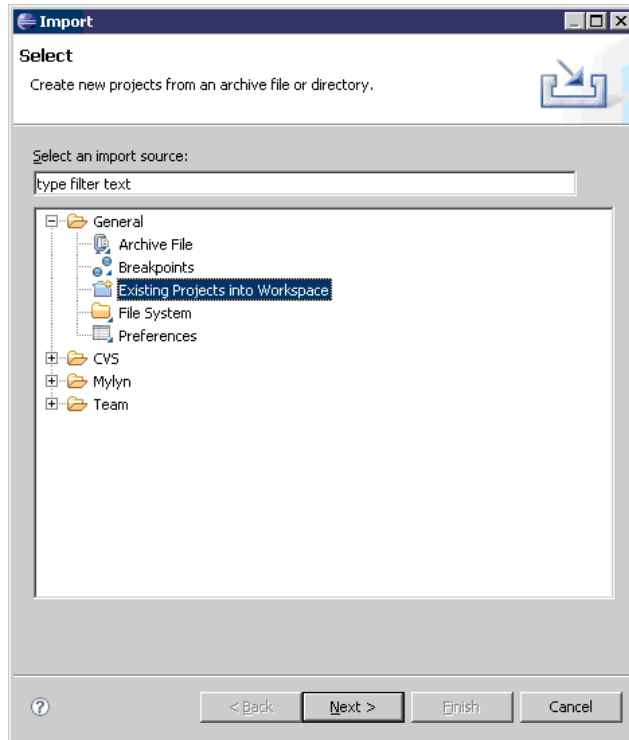
This task explains how to configure Eclipse so that you can run your Jython scripts in debug mode, thus enabling better visibility to job threads, trigger CIs, and results.

To configure Eclipse:

1 Prerequisites:

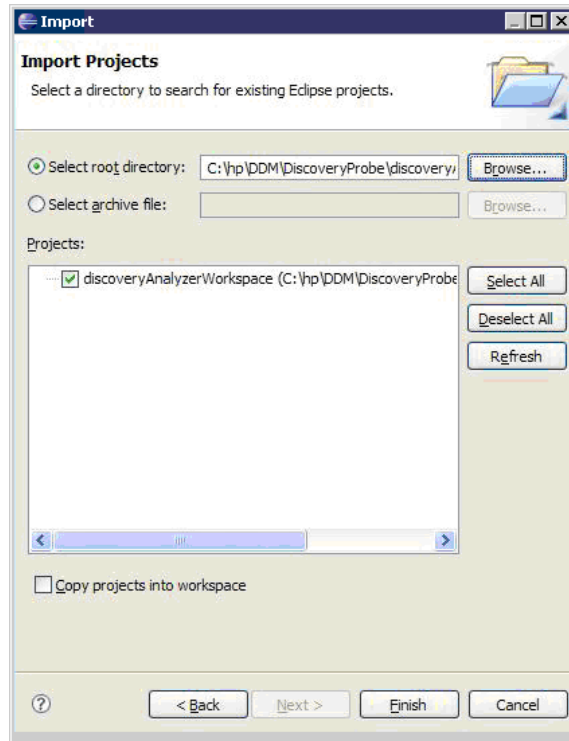
- ▶ Install Java 5.0 SE on your computer (if it is not already installed). The files are available on the Java Developers Web site <http://www.java.sun.com>.
- ▶ Install the latest Eclipse version on your computer. The application is available at www.eclipse.org.
- ▶ Install PyDev and PyDev Extensions, available at: <http://pydev.sourceforge.net/> (PyDev) and <http://www.fabioz.com/pydev/> (PyDev Extensions)
- ▶ The Discovery Probe must be installed on this computer.

2 Select File > Import > General > Existing Projects into Workspace:



**3 Click Next. Click Browse to select the following directory:
C:\hp\DDM\DiscoveryProbe\discoveryAnalyzerWorkspace:**

- 4 Verify that **discoveryAnalyzerWorkspace** is selected in the Projects pane and click **Finish**:

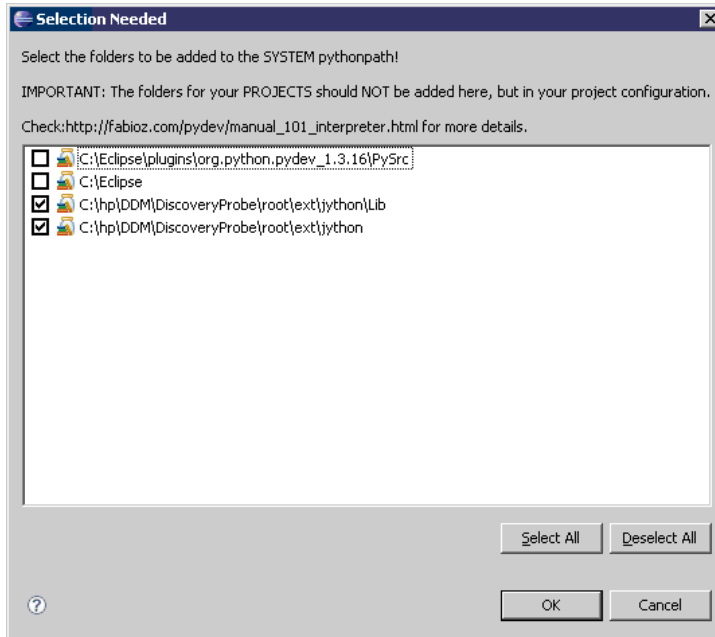


A discoveryAnalyzerWorkspace project is added to the Eclipse workspace.

- 5 Select **Window > Preferences** to open the **Preferences** dialog box. Select **Pydev > Interpreter - Jython > New**.

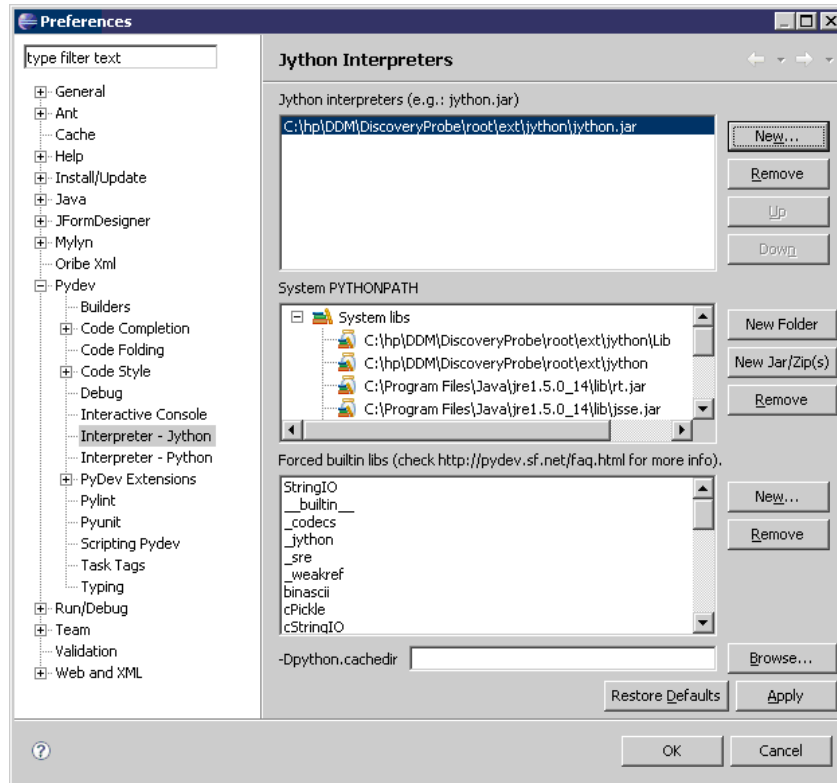
6 Select the following directories:

- C:\hp\DDM\DiscoveryProbe\root\ext\jython\
- C:\hp\DDM\DiscoveryProbe\root\ext\jython\lib



Click **OK**.

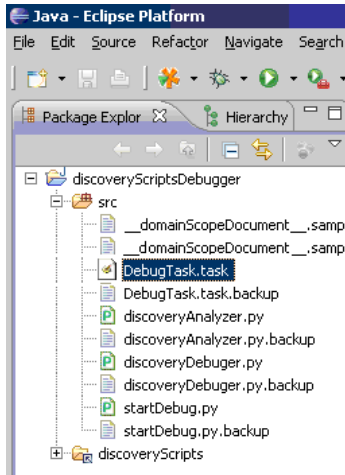
- 7 Return to the **Preferences** dialog box. Verify that the `jython.jar` file appears in the Jython interpreters list:



Click **OK**.

- 8 Select **Window > Open Perspective > Other** to open the **Open Perspective** dialog box. Select **PyDev**.
- 9 You can configure the job to be debugged either in a text editor or by using a graphic utility. For details on the text editor configuration, go to the next step. For details on the graphic utility configuration, skip to step 13 on page 78.

- 10 Select the **DebugTask.task** file in the **discoveryAnalyzerWorkspace\src** project:



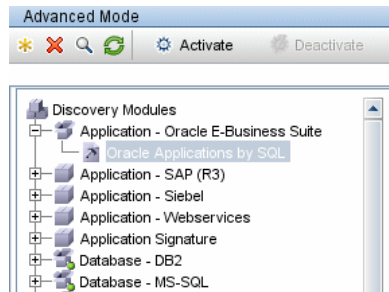
To edit this file in Eclipse, right-click **DebugTask.task** and select **Open With > Text Editor**.

The **DebugTask.task** file is a template you can use to debug your Jython scripts. It is located in the following directory:

C:\hp\DDM\DiscoveryProbe\discoveryAnalyzerWorkspace\src.

```
<taskInfo taskId="FOO Debug Task OOF">
  <jobId>Range IPs by ICMP</jobId>
  <destinationList>
    <destination>
      <destinationData name="domain">DefaultDomain</destinationData>
      <destinationData name="probeName">DefaultProbe</destinationData>
    </destination>
  </destinationList>
</taskInfo>
```

- 11** Change the `jobId` value to the name of the job to be debugged. For example, delete **Range IPs by ICMP** and enter **Oracle Applications by SQL**. You can copy the name from the job name in the Run Discovery page:



- 12** Add as many `destinationData` attributes as necessary (do not change the `destinationList` `className`). For example, to debug the **Oracle Applications by SQL** script, enter the following values:

```
<jobId>IIS Applications by NTCMD</jobId>
<destinationList
  className="com.hp.ucmdb.discovery.probe.tasks.BaseDestinationData">
  <destination>
    <destinationData name="fileToUpdate">discoveryJobs/Oracle Applications
    by SQL.xml</destinationData>
    <destinationData
      name="fileUpdateTime">1216821680058</destinationData>
    <destinationData
      name="id">bfd3d1ea8f7229e055c7004cf87ebd65</destinationData>
    <destinationData name="hostId" />
    <destinationData name="ip_address" />
    <destinationData name="ip_domain" />
  </destination>
</destinationList>
```

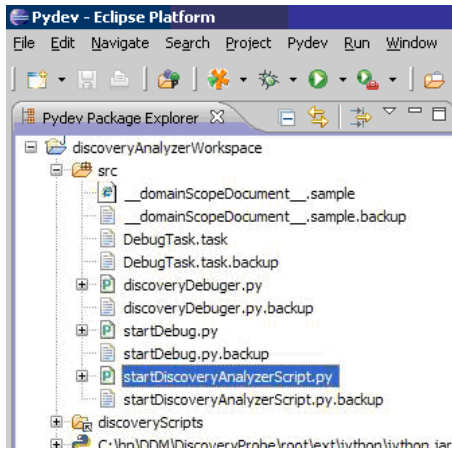
You can verify the exact `destinationData` attributes for each job by consulting the following log file:

C:\hp\DDM\DiscoveryProbe\root\logs\probeGW-taskResults.log.

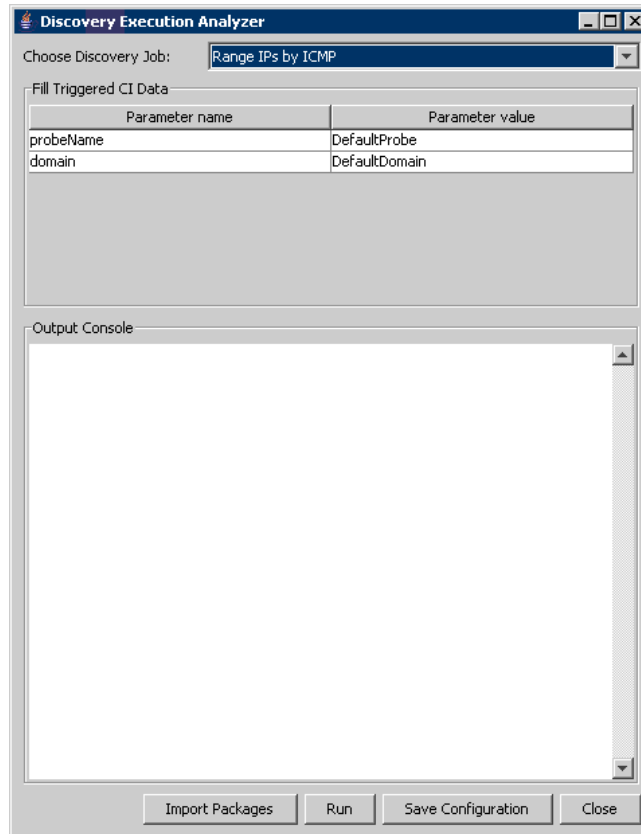
Note: Disregard UPDATE_SERVER_DATA jobs in the log file.

Skip to step 14 on page 79.

- 13** To use the graphic utility, select the **discoveryAnalyzer.py** file in the **discoveryAnalyzerWorkspace\src** project. Right-click the file and choose **Run as > Jython run**.



The Discovery Execution Analyzer dialog box is displayed:



Choose the Discovery Job, fill in the parameter values and save the configuration.

- 14** Add a breakpoint in the Jython script to be debugged.
- 15** In the `discoveryAnalyzerWorkspace` project, right-click `startDebug.py` and choose **Debug As > Jython Run**.

Troubleshooting and Limitations

The **pydev** plug-in including the Jython interpreter should be well configured. The recommended way is to select the Jython interpreter that is part of the Data Flow Probe installation: **Eclipse window > Preferences**. The workspace should be used that includes a **.pydevproject** defining the class path of **pydev**. It is selected automatically if the **pydev** is correctly installed.

3

Debugging Content With Discovery Analyzer

This chapter includes:

Concepts

- ▶ Using Discovery Analyzer to Debug Content Overview on page 81

Tasks

- ▶ Work with Discovery Analyzer on page 83

Reference

Troubleshooting and Limitations on page 89

Concepts

Using Discovery Analyzer to Debug Content Overview

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.

This section includes the following topics:

- ▶ "Tasks and Records" on page 82
- ▶ "Logs" on page 82

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. An execution 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.

Tasks

Work with Discovery Analyzer

The following procedure explains how to work with Discovery Analyzer.

This section includes the following topics:

- "Prerequisites" on page 83
- "Access Discovery Analyzer" on page 84
- "Define a Task" on page 85
- "Define a New Task" on page 85
- "Retrieve a Record" on page 86
- "Open a Task File" on page 86
- "Import a Job from the Database" on page 86
- "Edit a Task" on page 87
- "Save the Task and Logs" on page 87
- "Run the Task" on page 87
- "Send a Task Result to the Server" on page 88
- "Import Settings" on page 88
- "Breakpoints" on page 88

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 a UCMDB 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 88.

- ▶ The UCMDB server does not need to be installed.

2 Access Discovery Analyzer

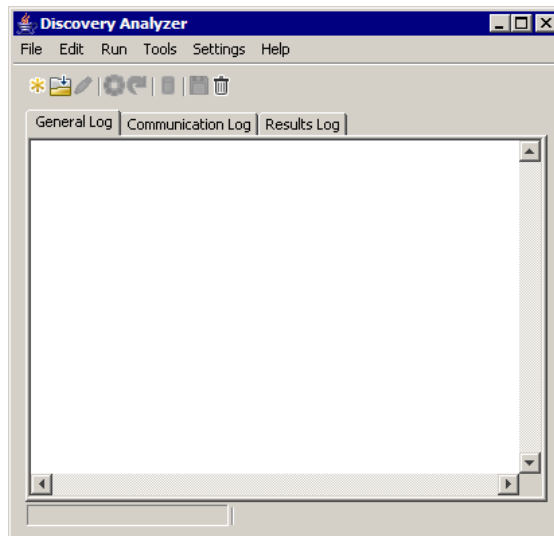
You access Discovery Analyzer either:

- ▶ When working with Eclipse. The recommended plug-in is **pydev**. Install PyDev and PyDev Extensions, available at: <http://pydev.sourceforge.net/> (PyDev) and <http://www.fabioz.com/pydev/> (PyDev Extensions).

The Probe installation comes with a default Eclipse workspace located at **C:\hp\DDM\DiscoveryProbe\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\DDM\DiscoveryProbe\root\lib\collectors\discoveryAnalyzer.cmd**. For details, see the following section.

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" on page 85.
- ▶ By importing a task from a record file. For details, see "Retrieve a Record" on page 86.
- ▶ By importing a saved task from a task file. For details, see "Open a Task File" on page 86.
- ▶ By retrieving a job from the Probe's internal database. For details, see "Import a Job from the Database" on page 86.

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.

Parameter	Value
ip_domain	
oid	
hostId	
ip_address	
credentialsId	
id	

- 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 Parameters Pane" in the *HP Universal CMDB 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 page 87.

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.

6 Open a Task File

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

7 Import a Job from the Database

You can retrieve a job 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 job to run and click **OK**.

- c Continue with the procedure for defining a task. For details, see "Save the Task and Logs" on page 87.

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 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**.

- b 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 82.
- c 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 UCMDB 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 a UCMDB server that receives tasks from the Probe that is installed on the same machine as Discovery Analyzer.

12 Import Settings

If the Probe has not yet run, you can import files needed by Discovery Analyzer. Access the Settings menu and import **domainScopeDocument.xml** or **domainScopeDocument.bin**. If you import the **.bin** file, you must import **key.bin** too. You can deploy any necessary packages (including scripts, adapters, and so on) by selecting: **Settings > Import packages**.

13 Breakpoints

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

Reference

Troubleshooting and Limitations

This section describes troubleshooting and limitations for the Discovery Analyzer.

- ▶ 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.
- ▶ The communication log is not supported from Eclipse.
- ▶ When executing the tool from Eclipse, the **DiscoveryProbe.properties** file must contain the following flag set to **true**:

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

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
```


4

Supporting Multi-Lingual Locales

This chapter includes:

Concepts

- ▶ Supporting Multi-Lingual Locales Overview on page 91
- ▶ Determining the Character Set for Encoding on page 92
- ▶ Resource Bundles on page 94

Tasks

- ▶ Add Support for a New Language on page 95
- ▶ Define a New Job to Operate With Localized Data on page 97
- ▶ Decode Commands Without a Keyword on page 98
- ▶ Change the Default Language on page 99
- ▶ Write Error Messages for a Specific Locale on page 99

Reference

- ▶ API Reference on page 100

Concepts

Supporting Multi-Lingual Locales Overview

Note: This functionality is available as part of Content Pack 3.00 or later.

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 UCMDB by the Schedule Manager.

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

Determining 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-Адрес** for the Russian OS, **IP-Adresse** for the German OS, and so on.

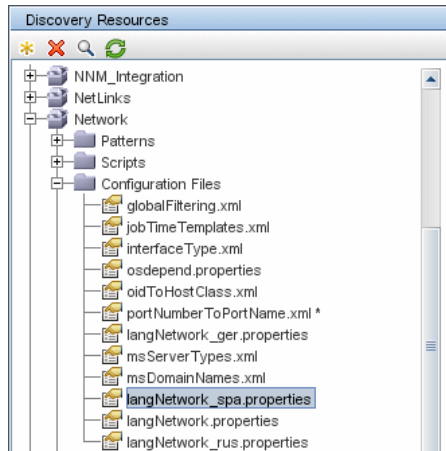
Once it is discovered in which language the command output is produced (# 1), possible character sets are limited to one or two (# 2). Furthermore, it is known which key words are contained in this output (# 5).

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.

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\DDM\DiscoveryProbe\root\lib\collectors\probeManager\discoveryConfigFiles**. They are downloaded from the UCMDB server as any other configuration file. They can be edited, added, or removed, in the Manage Discovery Resources window. For details, see "Configuration File Pane" in the *HP Universal CMDB 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.

Tasks

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)" on page 96
- "Declare and Register the Language Object" on page 96

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
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 "Resource Bundles" on page 94.

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 *HP Universal CMDB 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 100. 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)
```

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, you use the API for the **ShellUtils** class. For details, see "Data Flow Management Web Service API" on page 355.

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 step 4 on page 98 in "Define a New Job to Operate With Localized Data" on page 97.

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 step 3 on page 99. No additional decoding operations occur.

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 page 95.

Write Error Messages for a Specific Locale

This task includes the following steps:

- "Prerequisites" on page 99

1 Prerequisites

Prepare the package. For details, see "Create a Custom Package" in the *HP UCMDB Administration Guide*.

Reference

API Reference

This section includes:

- "The Language Class" on page 100
- "The executeCommandAndDecode Method" on page 101
- "The getCharsetName Method" on page 102
- "The useCharset Method" on page 102
- "The getLanguageBundle Method" on page 102
- "The osLanguage Field" on page 102

The Language Class

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

Fields

Name	Description
locale	Java object which represents locale.
bundlePostfix	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.
charsets	Character sets used to encode this language. Each language can have several character sets. For example, the Russian language is commonly encoded with the Cp866 and Windows-1251 encoding.

Name	Description
wmiCodes	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 OSLanguage section). One of the methods for identifying the OS language is to query the WMI class OS for the OSLanguage property.
codepage	Code page used with a specific language. For example, 866 is used for Russian machines and 437 for English machines. One of the methods for identifying the OS language is to retrieve its default codepage (for example, by the chcp 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 sudo 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 return 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.

Arguments

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

See also "The getCharsetName Method" on page 102.

The getLanguageBundle Method

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

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

Arguments

Name	Description
baseName	The name of the bundle without the language suffix, for example, langNetwork.
language	The language object. The ShellUtils.osLanguage 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.

5

Error Messages

This chapter includes:

Concepts

- ▶ Error Messages Overview on page 106

Tasks

- ▶ Write Error Messages on page 107

Reference

- ▶ Error-Writing Conventions on page 107
- ▶ Error Severity Levels on page 110

Troubleshooting and Limitations on page 111

Concepts

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. Users can drill down from the Trigger CI that caused the problem to view the error message itself.

DFM differentiates between errors that can 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.

All errors are saved to the `discovery_problems` table in the Probe Manager database schema. (The error information is saved to the database—and is not handled in the Probe's memory—to guarantee delivery to the server.) The Probe holds the latest list of problems for each Trigger CI. After each run, the Probe checks for changes and reports them in the Discovery Status pane.

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. For details, see "Write Error Messages" on page 107. Messages must conform to conventions, as described in "Error-Writing Conventions" on page 107.

DFM supports multi-language error messages. You can localize the messages you write so that they appear in the locale language. For details, see "Write Error Messages for a Specific Locale" on page 99.

See Also

- For details on searching for errors, see "Discovery Status Pane" in *HP Universal CMDB Data Flow Management Guide*.
- For details on setting communication logs, see "Execution Options Pane" in *HP Universal CMDB Data Flow Management Guide*.

Tasks

Write Error Messages

errors are reported with a message code id and their meaning is taken from a property file which is a deployable resource.

This task includes the following steps:

- "Prerequisites" on page 107

1 Prerequisites

Prepare the package. For details, see "Create a Custom Package" in *HP UCMDB Administration Guide*.

2 Prerequisites

Write a short and long version of the error message that is to appear
add message to appropriate file: explain about files

3 Provide a code for the message

Reference

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 between the number of arguments in the template and the actual number of parameters is assumed).

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

Property File Content

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

- a. `DDM_ERROR_MESSAGE_SHORT_45` = short error description
- b. `DDM_ERROR_MESSAGE_LONG_45` = long error description (might contain parameters, e.g. {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 (i.e. new message codes are added while old message codes are overridden).

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

Naming Conventions

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

where `xx` - stands for the locale (e.g. "`infraerr_fr.properties.errors`" or "`infraerr_en_us.properties.errors`").

Preserved error message codes

0-99 = numbers for infra

100 = content unclassified

101 - ??? = numbers for content

Unclassified Content Errors

1. In order to support old content without cause any regression, the UI and SDK's relevant methods handle differently errors of message code 100 (i.e. unclassified script error).
2. These errors will not be grouped (i.e. considered as error of the same type) by their message code but will be grouped by the content of the message. Meaning, if a script reports an error by the old deprecated methods (i.e. with a message string and without an error code), all messages will receive the same error code but in the UI or in SDK's relevant methods, different messages will be displayed as different errors.

Changes in framework

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

1. The following methods were added to the interface

```
"void reportError(int msgCode, String[] params);
```

```
"void reportWarning(int msgCode, String[] params);
```

```
"void reportFatal(int msgCode, String[] params);
```

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

```
"void reportError(String message);
```

```
"void reportWarning (String message);
```

```
"void reportFatal (String message);
```

Error Severity Levels

Every log file is set so that the information it records corresponds to a certain severity threshold. Because the various logs are used to keep track of different information, each is pre-set to an appropriate default level. For details on changing the default level, see "Changing Log Levels" in *HP UCMDB Administration Guide*.

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

- ▶ **Fatal.** This level reports serious errors such as a problem with the infrastructure, missing DLL files, or exceptions.
- ▶ **Debug.** This level is used by HP Software Support when troubleshooting problems.
- ▶ **Error.** 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).
- ▶ **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 or remote host, or that invalid data caused an attribute not to be properly calculated.
- ▶ **Info.** The log records all activity. Most of the information is normally routine and of little use and the log file quickly fills up.
- ▶ **Success.** The Trigger CI ran successfully.

Note: The names of the different log levels may vary slightly on different servers and for different procedures. For example, **Info** may be referred to as **Always logged** or **Flow**.

Troubleshooting and Limitations

- ▶ For Fatal errors, you should contact HP Software Support.

For other errors, check the CIs. For example, a Trigger CI that does not fall within the Probe's range may show an error.

6

Integration Framework SDK

This chapter includes:

Concepts

- ▶ Federation Framework – Overview on page 114
- ▶ Adapter and Mapping Interaction with the Federation Framework on page 118
- ▶ Federation Framework Flow for FTQL on page 119
- ▶ Federation Framework Flow for Replication on page 131
- ▶ Adapter Interfaces on page 133

Tasks

- ▶ Add an Adapter for a New External Data Source on page 134
- ▶ Implement the Default Mapping Engine on page 141
- ▶ Add a New Adapter – Scenario on page 142

Reference

- ▶ Adapter Capabilities on page 148

Concepts

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 *HP UCMDB Administration Guide*.
-

The Federation Framework functionality uses an API to retrieve information from federated sources. The Federation Framework provides two main capabilities:

- **Federation on the fly.** All queries are run over original data stores and results are built on the fly in the CMDB.
- **Data Replication.** Replicates data (topological data and CI properties) from one data store to another.

Both action types require an adapter for each data store, which can provide the specific capabilities of the data store and retrieve and/or update the required data. Every request to the data store is made through its adapter.

This section also includes the following topics:

- "Federation on the Fly" on page 115
- "Data Replication" on page 117

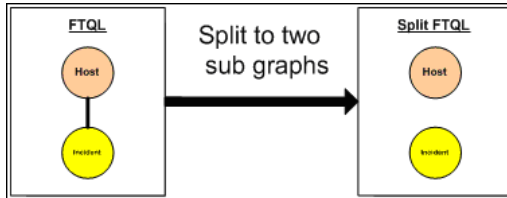
Federation on the Fly

Federated TQL enables data retrieval from any external data repository without replicating its data.

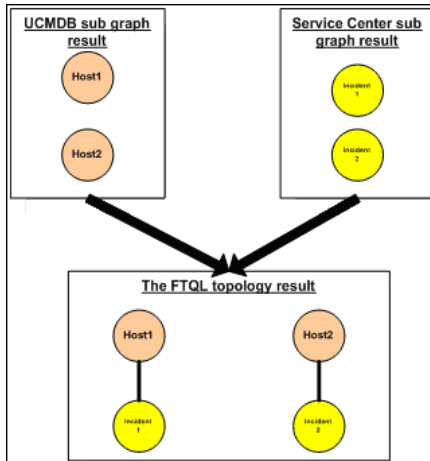
A federated TQL query uses adapters that represent external data stores, to create appropriate external relationships between CIs from different external data repositories and the UCMDB CIs.

Example of Federation-on-the-Fly Flow:

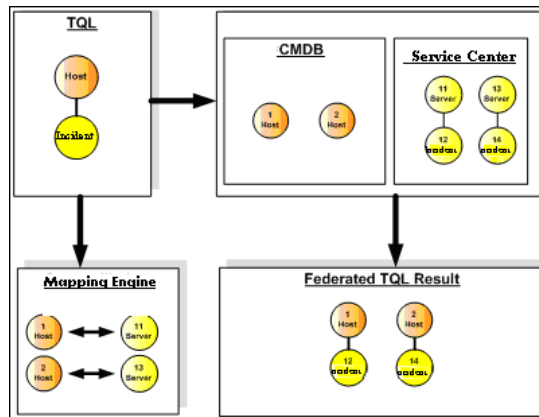
- 1 The Federation Framework splits a Federated TQL (FTQL) into several subgraphs, where all nodes in a subgraph refer to the same data store. Each subgraph is connected to the other subgraphs by a virtual relationship (but itself contains no virtual relationships).



- 2 After the FTQL 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 FTQL topology is calculated, the Federation Framework retrieves a layout for the topology result.



Data Replication

You replicate data if you have several data stores with a large amount of data, and another data store that uses a view with data from these data stores.

In data replication, data stores are divided into two categories: source and target. Data is retrieved from the source data store and updated to the target data store. The replication is based on query names, that is, data is synchronized between the source data store and target data store and is retrieved by a query name in the source data store. For example, in UCMDB, the query name is the name of the TQL. However, in another data store the query name can be a code name that returns data. The adapter is designed to correctly handle the query name.

Each query name in the source data store can be defined as an exclusive query. This means that the CIs and relationships in the query results are unique in the target data store, and no other query can bring them to the target. The adapter of the source data store supports specific queries, and can retrieve the data from this data store. The adapter of the target data store enables the update of retrieved data on this data store.

The replication process flow includes the following steps:

- 1** Retrieving the topology result with signatures from the source data store.
- 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 store with the received full layout of CIs and relationships. If any CIs or relationships are deleted in the source data store and the query is exclusive, the replication process removes the CIs or relationships in the target data store as well.

Adapter and Mapping Interaction with the Federation Framework

An adapter is an entity in UCMDB that represents external data (data that is not saved in UCMDB). 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 FTQL.

This section also includes the following topics:

- "Adapter Lifecycle" on page 119
- "Adapter assist Methods" on page 119

Adapter Lifecycle

An adapter instance is created for each external data store. 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 store configuration, logger, and so on. The adapter lifecycle ends when the data store 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 store if it is required.

Adapter assist Methods

The adapter has several `assist` methods that can add external data store 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 store for a given configuration.
- The second method is relevant only for the source adapter and returns the supported queries for replication.
- The third method is relevant only for FTQL and returns supported external classes by the external data store.

All these methods are used when you create new data store configurations.

Federation Framework Flow for FTQL

This section includes the following topics:

- "Definitions and Terms" on page 120
- "Mapping Engine" on page 121
- "FTQL Adapter" on page 121
- "Flow Diagrams" on page 121

Definitions and Terms

Reconciliation data. The rule for matching CIs of the specified type that are received from the CMDB and the external data store. The reconciliation rule can be of three types:

- ▶ **ID reconciliation.** This can be used only if the external data store contains the CMDB 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 host type by the `ip_address` property that belongs to the `ip` 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 CMDB CIs.

Reconciliation CI type. The type of CIs that represent reconciliation objects. These CIs must be stored in both the CMDB and in the external data stores.

Mapping engine. A component that identifies relations between CIs from different data stores that have a virtual relationship between them. The identification is performed by reconciling CMDB reconciliation objects and external CI reconciliation objects.

Mapping Engine

Federation Framework uses the Mapping Engine to calculate the FTQL. The Mapping Engine connects between CIs that are received from different data stores 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 CMDB. This end is a reconciliation type. For the calculation of the two subgraphs, a virtual relationship can start from any end node.

FTQL Adapter

The FTQL adapter brings two kinds of data from external data stores: external CI data and reconciliation objects that belong to external CIs.

External CI data. The external data that does not exist in the CMDB. It is the target data of the external data store.

Reconciliation object data. The auxiliary data that is used by the federation framework to connect between CMDB 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 whose data is retrieved from . Reconciliation objects should fit the adapter received to reconciliation data. The reconciliation object can be one of three types: `IdReconciliationObject`, `PropertyReconciliationObject`, or `TopologyReconciliationObject`.

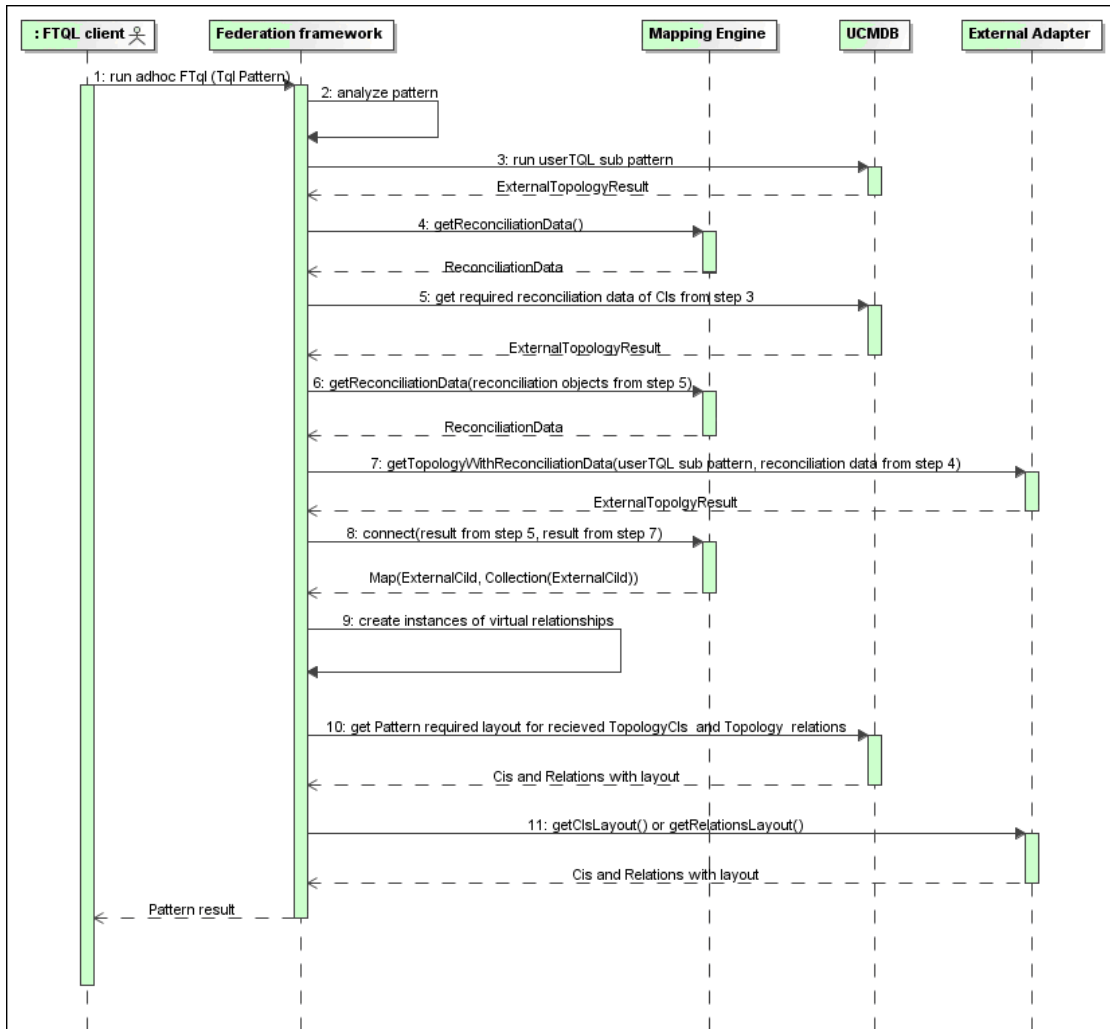
Flow Diagrams

The following diagrams illustrate the interactions between the Federation Framework, UCMDB, the adapter, and the Mapping Engine. The FTQL in the example diagrams has only one virtual relationship, so that only UCMDB and one external data store are involved in the FTQL.

In the first diagram the calculation begins in UCMDB 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.

The Calculation Starts at the HP Universal CMDB End

The following sequence diagram illustrates the interaction between the Federation Framework, UCMDB, the adapter, and the Mapping Engine. The FTQL in the example diagram has only one virtual relationship, so that only UCMDB and one external data store are involved in the FTQL.

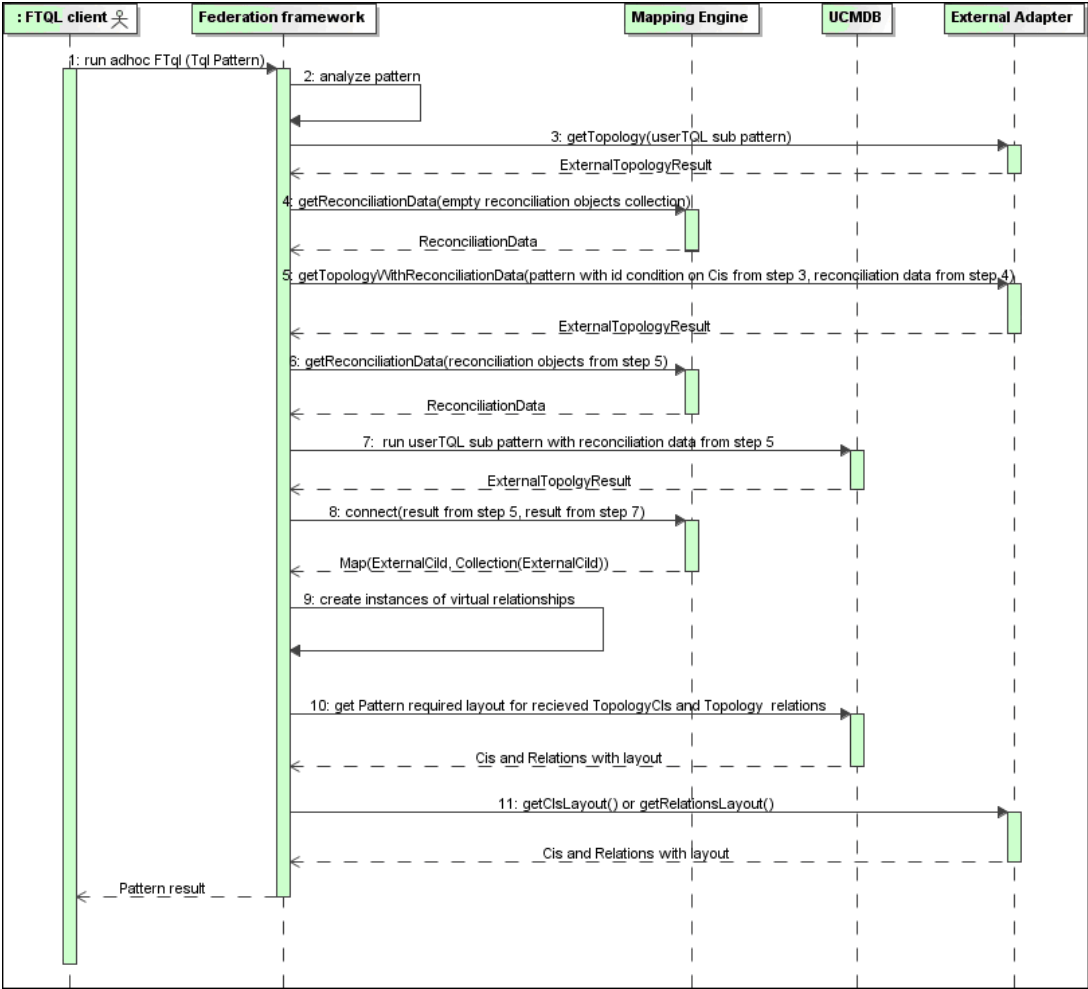


The numbers in this image are explained below:

Number	Explanation
1	The Federation Framework receives a call for a FTQL calculation.
2	The Federation Framework analyzes the adapter, finds the virtual relationship, and divides the original TQL into two sub-adapters—one for UCMDB and one for the external data store.
3	The Federation Framework requests the topology of the sub-TQL from UCMDB.
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 UCMDB to the external data store. 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 host CIs, the IP address of IP is required too).
5	The Federation Framework requests reconciliation data for the CIs of the virtual relationship ends that were received in step 3 from UCMDB.
6	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.
7	The Federation Framework requests the topology of the sub-TQL from the external data store. The external adapter receives the reconciliation data from step 6 as a parameter.

Number	Explanation
8	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 UCMDB in step 5 and the <code>secondResult</code> parameter is the external topology result received from the External Adapter in step 7. The Mapping Engine returns a map where External CI ID from the first data store (UCMDB in this case) is mapped to the External CI IDs from the second (external) data store.
9	For each mapping, the Federation Framework creates a virtual relationship.
10	After the calculation of the FTQL results (only at the topology stage), the Federation Framework retrieves the original TQL layout for the resulting CIs and relationships from the appropriate data stores.

The Calculation Starts at the External Adapter End



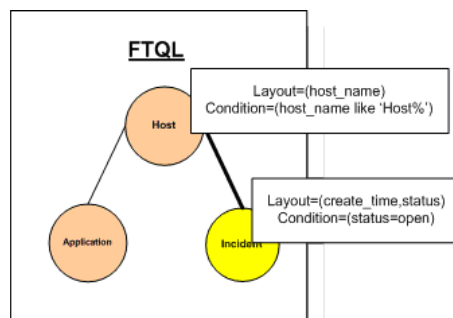
The numbers in this image are explained below:

Number	Explanation
1	The Federation Framework receives a call for an FTQL calculation.
2	The Federation Framework analyzes the adapter, finds the virtual relationship, and divides the original TQL into two sub-adapters – one for UCMDB and one for the external data store.
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.
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 UCMDB to the external data store. 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 host CIs, the IP address of IP is required too).
5	The Federation Framework requests reconciliation objects for the CIs that were received in step 3 from the external data store. The Federation Framework calls the <code>getTopologyWithReconciliationData()</code> 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.
6	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.

Number	Explanation
7	The Federation Framework requests the topology of the sub-TQL with reconciliation data from step 6 from UCMDB.
8	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 UCMDB at step 7. The Mapping Engine returns a map where the External CI ID from the first data store (the external data store in this case) is mapped to the External CI IDs from the second data store (UCMDB).
9	For each mapping, the Federation Framework creates a virtual relationship.
10	After the calculation of the FTQL results (only at the topology stage), the Federation Framework retrieves the original TQL layout for the resulting CIs and relationships from the appropriate data stores.

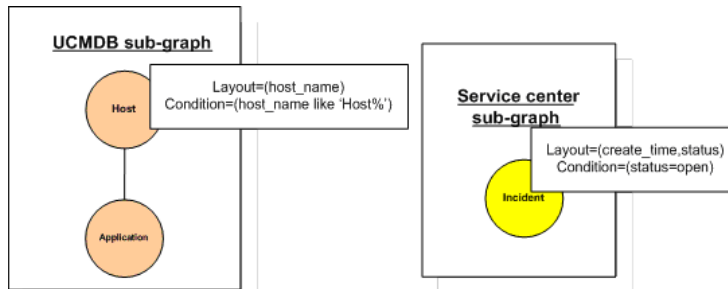
Example of Federation Framework Flow for FTQL

This example explains how to view all open incidents on specific hosts. The ServiceCenter data store is the external data store. The host instances are stored in UCMDB, and the incident instances are stored in ServiceCenter. It is assumed that to connect the incident instances to the appropriate host, the `host_name` and `ip_address` properties of the host and IP are needed. These are reconciliation properties that identify the hosts from ServiceCenter in UCMDB.

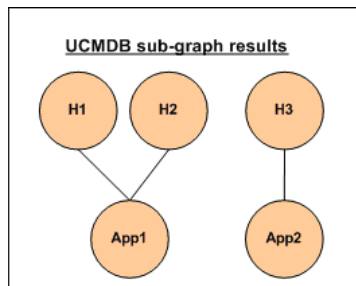


Note: For attribute federation, the adapter's **getTopology** method is called. The reconciliation data is adapted in the user TQL (in this case, the CI element).

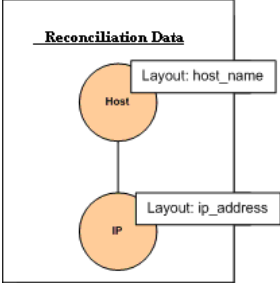
- 1 After analyzing the adapter, the Federation Framework recognizes the virtual relationship between Host and Incident and splits the FTQL into two subgraphs:



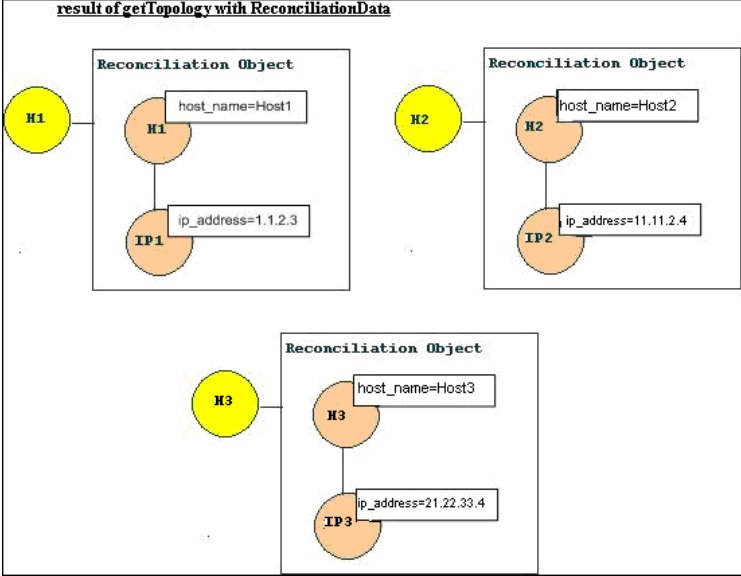
- 2 The Federation Framework runs the the UCMDB subgraph to request the topology, and receives the following results:



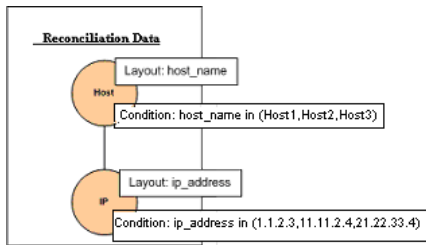
- 3 The Federation Framework requests, from the appropriate Mapping Engine, the reconciliation data for the first data store (UCMDB) that contains the information to connect between received data from two data stores. The reconciliation data in this case is:



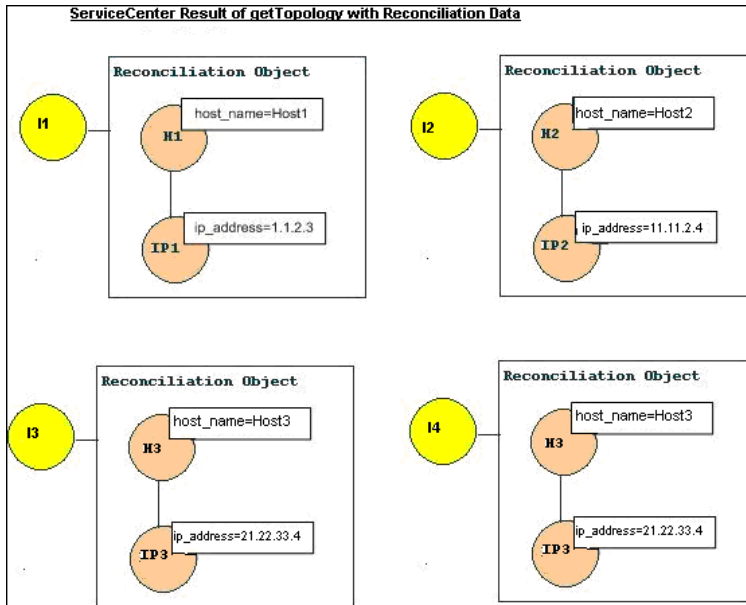
- 4 The Federation Framework creates a one-node topology query with Host node and ID conditions on it from the previous result (host_id in H1, H2, H3), and runs this query with the required reconciliation data on UCMDB. The result includes Host CIs that are relevant to the ID condition and the appropriate reconciliation object for each CI:



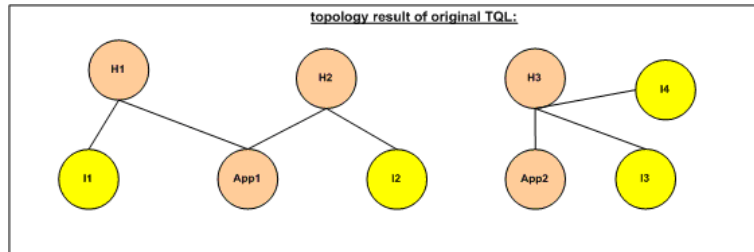
- The reconciliation data for ServiceCenter should contain a condition for `host_name` and `ip_address` that is derived from the reconciliation objects received from UCMDB:



- 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 UCMDB and ServiceCenter.

Federation Framework Flow for Replication

This section includes the following topics:

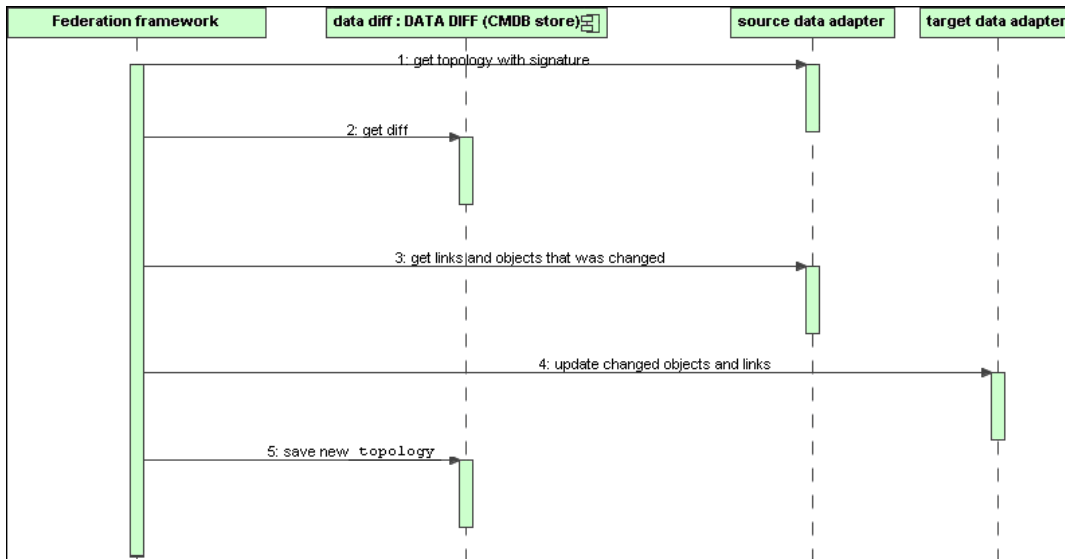
- "Definitions and Terms" on page 131
- "Flow Diagram" on page 132

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 replication 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 store. 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 store. 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" on page 133
- "Adapter Interfaces for FTQL" on page 133

Definitions and Terms

The external relation. The relation between two external CI types that are supported by the same adapter.

Adapter Interfaces for FTQL

Use the appropriate adapter interface for each adapter, as follows.

A **oneNode 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 **PatternTopology** interface.

A **Pattern topology interface** is used when the adapter supports more than one external CI type and supports at least one relation type between supported external CI types. That is, the adapter should implement the **Pattern Topology** interface if it intends receiving queries for external data with relations.

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 stores is calculated by the ID.
- **OneNodeTopologyPropertyReconciliationDataAdapter.** Use if the adapter supports a **single-node TQL** and the reconciliation between data stores is done by the properties of one CI.
- **OneNodeTopologyDataAdapter.** Use if the adapter supports a **single-node TQL** and the reconciliation between data stores is done by topology.

Pattern Topology Interfaces

The following interfaces have different types of reconciliation data:

- **PatternTopologyIdReconciliationDataAdapter**. Use if the adapter supports a **complex TQL** and the reconciliation between data stores is done by the ID.
- **PatternTopologyPropertyReconciliationDataAdapter**. Use if the adapter supports a **complex TQL** and the reconciliation between data stores is done by single-node properties.
- **PatternTopologyDataAdapter**. Use if the adapter supports a **complex TQL** and the reconciliation between data stores is done by topology.

Additional Interfaces

- **SortResultDataAdapter**. Use if you can sort the resulting CIs in the external data store.
- **FunctionalLayoutDataAdapter**. Use if you can calculate the functional layout in the external data store.

Adapter Interfaces for Replication

- **SourceDataAdapter**. Use for source adapters in replication jobs.
- **TargetDataAdapter**. Use for target adapters in replication jobs.

Tasks



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" on page 135
- "Define Valid Relationships for Virtual Relationships" on page 135
- "Define an Adapter Configuration" on page 137

- "Implement the Adapter" on page 138
- "Define Reconciliation Rules or Implement the Mapping Engine" on page 139
- "Add Jars Required for Implementation to the Class Path" on page 139
- "Deploy the Adapter" on page 139
- "Redeploy the Adapter" on page 140

1 Prerequisites

Model-supported adapter classes for CIs and relationships in the UCMDB Data Model: as an adapter developer, you should:

- have knowledge of the hierarchy of the UCMDB CI types to understand how external CITs are related to the UCMDB CITs
- model the external CITs in the UCMDB class model
- add the definitions for new CI types and their relationships
- define valid relationships in the UCMDB class model for the valid relationships between adapter inner classes. (The CITs can be placed at any level of the UCMDB 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 UCMDB 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 the FTQL adapter.

Determine the relationship between your data and the UCMDB data, that is, define your virtual relationships. You can do this by adding a valid relationship, where one end of the relationship does not refer to your adapter supported classes. A valid relationship for a virtual relationship is different from a regular valid relationship only in that it has a qualifier that defines a Mapping Engine class implementation.

Example of Valid Relationship Definition:

In the following example of a valid relationship definition, the relation of type `history_link` between instances of type `it_world` to instances of type `HistoryChange` is valid. To connect between instances of these types the Federation Framework should use the `HistoryMappingEngine` implementation class:

```
<Valid-Link>
  <Class-Ref class-name="history_link" />
  <End1 class-name="it_world" />
  <End2 class-name="HistoryChange" />
  <Valid-Link-Qualifiers>
    <Valid-Link-Qualifier name="EXTENDED_VALID_LINK">
      <Data-Items>
        <Data-Item name="mapping_engine_class" type="string">
com.mercury.topaz.adapters.CmdbHistoryAdapter.HistoryMappingEngine
        </Data-Item>
      </Data-Items>
    </Valid-Link-Qualifier>
  </Valid-Link-Qualifiers>
</Valid-Link>
```

If you have only one Mapping Engine implementation for all virtual relationships, you can define it in the adapter configuration and not as a qualifier in the valid relationship.

3 Define an Adapter Configuration

Add an XML adapter configuration file that contains the following details:

- **Adapter ID.** A unique ID for the adapter.
- **Adapter name.** A fully-qualified Java implementation class name for the adapter.
- **Adapter capabilities.** Defines the capabilities the adapter supports. For details, see "Adapter Capabilities" on page 148.
- **Fields to connect.** Defines the fields that the user must supply to connect to the external data store.
- **Default mapping engine.** Defines the default mapping engine for virtual relationships. Relevant only for FTQL-supported adapters.

See the schema folder for the schema of the adapter configuration.

Example of Adapter Configuration Definition:

```
<adapter-config adapter-id="CmdbRmiAdapter">
  <class-name>com.mercury.topaz.adapters.cmdb.CmdbRmiAdapter</class-
name>
  <adapter-capabilities>
    <support-federated-query>
      <supported-classes>
        <supported-class is-derived="true" all-attributes-supported="true"
name="hostresource"/>
      </supported-classes>
      <topology>
        <pattern-topology>
          <functional-layout/>
        </pattern-topology>
        <advanced-capabilities>
          <calculated-attribute></calculated-attribute>
        </advanced-capabilities>
      </topology>
      <result>
        <sort-result/>
      </result>
    </support-federated-query>
    <support-replicatioin-data>
      <source/>
      <target/>
    </support-replicatioin-data>
  </adapter-capabilities>
  <fields-to-connect>
    <field>host</field>
    <field>customerId</field>
  </fields-to-connect>
  <default-mapping-
engine>com.mercury.topaz.adapters.cmdb.CmdbMappingEngine</default-mapping-
engine>
</adapter-config>
```

4 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.

5 Define Reconciliation Rules or Implement the Mapping Engine

If your adapter supports a federated query (FTQL), you should define the default Mapping Engine to use for reconciliation. (From version 8.00, the default mapping engine implementation is supported.)

You can use this implementation to define reconciliation rules only in the configuration file. For details, see "Create the reconciliation_rules.txt File" on page 147. For a special implementation, you should use the implementation class that implements the Mapping Engine interface.

6 Add Jars Required for Implementation to the Class Path

To implement your classes, add the **federation_api.jar** file to your class path.

7 Deploy the Adapter

- a Deploy the adapter package. For general details on deploying a package, see "Package Manager" in the *HP UCMDB 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 UCMDB.

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 virtual relationship is defined as a valid relationship and can include a data item named `mapping_engine_class`, which defines the mapping engine fully-qualified class name.

- The adapter configuration definition XML file should be located in the adapter folder in the package.
- **Descriptor.** Defines the package definitions.

b Deploy your code:

- Create a class that implements all required adapter interfaces.
- Implement a mapping engine (if you are writing a federated query adapter).
- Place your compiled classes (normally a jar file) together with all your based-on *.jar files in the `C:\hp\UCMDBServer\j2f\fcmdb\CodeBase\<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 also use the `C:\hp\UCMDBServer\j2f\fcmdb\CodeBase\<adapter id>` folder as your root folder.

8 Redeploy the Adapter

The adapter definitions and implementation may become altered as a result of external class definitions, changes in adapter capabilities, or changes in implementation. If this happens, redeploy the definitions or implementations.

a Redeploy an adapter package.

If your external classes definition or adapter definition was altered, create an updated adapter package and redeploy it using the package mechanism. For details, see "Deploy a Package" in the *HP UCMDB Administration Guide*.

b Redeploy your code.

If your implementation code or private configuration altered, do the following:

- ▶ Create updated *.jar or configuration files, and place them in the `C:\hp\UCMDBServer\j2f\fcmdb\CodeBase\` folder.
- ▶ Call the next JMX method with the appropriate customer ID and adapter ID:
`FCmdb Config Services > loadOrReloadCodeBaseForAdapterId.`

Implement the Default Mapping Engine

To use the default Mapping Engine implementation you should define it as the default Mapping Engine in the `adapter_config.xml` file and create a `reconciliation_rules.txt` file in the `<adapter-id>/META-INF` folder.

This task includes the following steps:

- ▶ "Add the Default Implementation to the Configuration File" on page 141
- ▶ "Configure the `reconciliation_rules.txt` File" on page 141

1 Add the Default Implementation to the Configuration File

To add the default Mapping Engine implementation to the `adapter_config.xml` file, add the following element:

```
<default-mappingengine>  
com.mercury.topaz.cmdb.shared.fcmdb.ftql.mappingEngine.AdapterMappingEngine  
</default-mappingengine>
```

2 Configure the `reconciliation_rules.txt` File

This file is used to configure the reconciliation rules. Each row in the file represents a rule. For example:

```
reconciliation_type[host] expression[^host.host_hostname OR ip.ip_address]  
end1_type[host] end2_type[ip] link_type[contained]
```

The **reconciliation_type** parameter is filled with the type of CI on which the reconciliation is performed (the UCMDB 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 UCMDB 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 UCMDB ID, use **cmdb_id** as the attribute name in expression.

Add a New Adapter – Scenario

This example illustrates how to add an adapter for ServiceCenter.

This task includes the following steps:

- "Create a CIT Definition" on page 143
- "Relate the CITs to the ServiceCenter Entity" on page 143

- "Define the Adapter Configuration XML File" on page 143
- "Create the reconciliation_rules.txt File" on page 147
- "Write Implementations for the ServiceCenter Adapter and Mapping Engine" on page 148

1 Create a CIT Definition

The relevant ServiceCenter entities are Incident, Problem, and RFC. For each entity, create the CIT definition. Decide where to place these definitions in the UCMDB class model hierarchy. The entities refer to IT processes, so it makes sense to put them under the IT Process CIT. RFC refers to IT change, so put the RFC class definition under the IT Change CIT. There is no relationship between Incident, Problem, and RFC CITs, so do not add definitions for such relationships.

2 Relate the CITs to the ServiceCenter Entity

Determine to which CITs in the UCMDB you want to relate the ServiceCenter entities. For the purpose of this example, the only relevant entity is Host. Add a valid relationship for each of the following pairs: [Host, Incident], [Host, Problem], [Host, RFC]. Each valid relationship should have a qualifier with the Mapping Engine implementation class name that supports connection between the ends of this relationship. It can be the same Java class for all three cases or three different classes, depending on your implementation.

3 Define the Adapter Configuration XML File

- Define the adapter ID.** For this example, assume the ID is ServiceCenterAdapter and the class implementation is adapter.ServiceCenterAdapterImpl. The beginning of the configuration adapter file is:

```
<adapter-config adapter-id=" ServiceCenterAdapter ">  
  <class-name> adapter.ServiceCenterAdapterImpl </class-name>
```

b Define adapter capabilities.

- ▶ In this example, the adapter supports a federation query but not replication data. Therefore, its capabilities XML should contain the support-federated-query element but not support-replication-data.
- ▶ Add the supported classes with supported attribute conditions. Suppose all three classes have property status and the system supports only the equal condition for this attribute. The supported_classes element has the following definition:

```
<supported-classes>
  <supported-class is-derived="true" all-attributes-supported="false"
name="incident">
    <supported-conditions>
      <attribute-operators attribute-name="status">
        <operator> EQUEL </operator>
      </attribute-operators>
    </supported-conditions>
  </supported-class>
  <supported-class is-derived="true" all-attributes-supported="false"
name="change">
    <supported-conditions>
      <attribute-operators attribute-name="status">
        <operator> EQUEL </operator>
      </attribute-operators>
    </supported-conditions>
  </supported-class>
  <supported-class is-derived="true" all-attributes-supported="false"
name="problem">
    <supported-conditions>
      <attribute-operators attribute-name="status">
        <operator>EQUEL</operator>
      </attribute-operators>
    </supported-conditions>
  </supported-class>
</supported-classes>
```


- Define the topology that the ServiceCenter adapter supports. Because there are no relationships between the ServiceCenter entities, it is sufficient that the ServiceCenter adapter support single-node topology. For simplicity's sake, assume that the ServiceCenter adapter has no advanced capabilities and cannot calculate the value of calculated attributes. The topology element definition is as follows:

```
<topology>
  <one-node-topology/ >
</topology>
```

- For simplicity's sake, assume that ServiceCenter does not support sorting. Therefore, the capability should not contain the element result.
- c Add the element that defines the information required for the ServiceCenter adapter to connect to ServiceCenter.** Assume this includes host, port, and user. Because the password for ServiceCenter can be empty, do not define this field as required. However, the ServiceCenter adapter checks this field, and if a value exists, it uses the password value to connect. The fields-to-connect element has the following definition:

```
<fields-to-connect>
  <field>host</field>
  <field>port</field>
  <field>user</field>
</fields-to-connect>
```

- d Add the default Mapping Engine element.** The following example uses the default implementation Mapping Engine, which takes the following definition:

```
<default-mappingengine>
  com.mercury.topaz.cmdb.shared.fcldb.ftql.mappingEngine.
AdapterMappingEngine
</default-mappingengine>
```

The configuration XML definition of the Service Desk adapter is complete. The following is the finished definition:

```
<adapter-config adapter-id=" ServiceCenterAdapter ">
  <class-name> adapter.ServiceCenterAdapterImpl </class-name>
  <adapter-capabilities>
    <support-federated-query>
      <supported-classes>
        <supported-class is-derived="true" all-attributesupported="false"
name="incident">
          <supported-conditions>
            <attribute-operators attribute-name="status">
              <operator> EQUAL </operator>
            </attribute-operators>
          </supported-conditions>
        </supported-class>
        <supported-class is-derived="true" all-attributesupported="false"
name=" change ">
          <supported-conditions>
            <attribute-operators attribute-name="status">
              <operator> EQUAL </operator>
            </attribute-operators>
          </supported-conditions>
        </supported-class>
        <supported-class is-derived="true" all-attributesupported="false"
name=" problem ">
          <supported-conditions>
            <attribute-operators attribute-name="status">
              <operator> EQUAL </operator>
            </attribute-operators>
          </supported-conditions>
        </supported-class>
      </supported-classes>
      <topology>
        <one-node-topolgy/>
      </topology>
    </support-federated-query>
  </adapter-capabilities>
```

```

        <fields-to-connect>
            <field>host</field>
            <field>port</field>
            <field>user</field>
            <field>password</field>
        </fields-to-connect>
        <default-mappingengine>
            com.mercury.topaz.cmdb.shared.fcmdb.ftql.mappingEngine.
AdapterMappingEngine
        </default-mappingengine>
    </adapter-config>

```

4 Create the reconciliation_rules.txt File

You can add a reconciliation rule for a host CIT only. This is because only host CITs have valid relationships with external CITs. For example, a host CI in UCMDB is matched to a host CI in ServiceCenter through the `host.host_hostname` attribute or through the `ip.ip_address` 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 `host.host_hostname` attribute is equal, the rule matches the hosts.
- If the `host.host_hostname` attribute is not equal, the rule does not match the hosts.
- If the `host.host_hostname` attribute is null in one of the compared CIs, the rule checks the `ip.ip_address` attribute. If the `ip.ip_address` attribute is equal, the rule matches the hosts.

The rule definition in the `reconciliation_rules.txt` file is as follows:

```

reconciliation_type[host] ordered expression[host.host_hostname or ip.ip_address]
end1_type[host] end2_type[ip] link_type[contained]

```

5 Write Implementations for the ServiceCenter Adapter and Mapping Engine

The Mapping Engine implementation should implement the MappingEngine interface. The ServiceCenter adapter should implement the OneNodeTopologyDataAdapter interface according to its capabilities and the fact that reconciliation is performed by a topology rule.

Reference

Adapter Capabilities

This section describes adapter capabilities:

- **Support federated query.** Included in adapter capabilities if the adapter implementation supports FTQL. Federated query capabilities include:
 - **Supported classes.** Defines the supported classes in the federated query. The supported class definition includes supported class name, whether derived classes of this class are also supported, and the possible condition operations for class attributes. The possible values for condition operators are:
 - IS_NULL
 - EQUALS
 - NOT_EQUALS
 - GREATER
 - GREATER_OR_EQUAL
 - LESS
 - LESS_OR_EQUAL
 - IN
 - EQUALS_CASE_INSENSITIVE
 - LIKE
 - LIKE_CASE_INSENSITIVE

- CHANGED_DURING
- UNCHANGED_DURING

If derived classes are supported, the condition operation for class attributes is also derived. You can ignore definition of supported classes in adapter capabilities configuration by implementing the **getSupportedClasses** method of the `FTqlDataAdapter` interface. This might be useful if your supported classes or supported attribute condition operators can be changed dynamically.

- **Topology.** Defines whether adapter topology or single-node topology is supported, and the federated TQL advanced capabilities.
- **Result.** Defines whether the adapter can sort the resulting CIs.
- **Support replication data.** Included in adapter capabilities if the adapter implementation supports data replication. Next capabilities are part of replication capabilities:
- **Source.** Defines that the adapter can be used in a replication job as source.
- **Target.** Defines that the adapter can be used in a replication job as target.

7

Generic Database Adapter

This chapter includes:

Concepts

- ▶ Generic Database Adapter Overview on page 153
- ▶ Non-supported TQL Queries on page 153
- ▶ Reconciliation on page 154
- ▶ Hibernate as JPA Provider on page 155

Tasks

- ▶ Validate Adapter Setup on page 157
- ▶ Prepare the Adapter Package on page 162
- ▶ Configure the Adapter on page 164
- ▶ Deploy the Adapter on page 172
- ▶ Edit the Adapter on page 173
- ▶ Load the Adapter on page 173
- ▶ Create an Integration Point on page 174
- ▶ Create a View on page 174
- ▶ Calculate the Results on page 176
- ▶ View the Results on page 177
- ▶ View Reports on page 178
- ▶ Enable Log Files on page 179

Reference

- ▶ Federated Database Configuration Files on page 179

- ▶ The adapter.conf File on page 180
 - ▶ The simplifiedConfiguration.xml File on page 181
 - ▶ The orm.xml File on page 183
 - ▶ The reconciliation_types.txt file on page 189
 - ▶ The reconciliation_rules.txt File (for backwards compatibility) on page 189
 - ▶ The transformations.txt File on page 192
 - ▶ The persistence.xml File on page 193
 - ▶ The discriminator.properties File on page 194
 - ▶ The replication_config.txt File on page 196
 - ▶ The fixed_values.txt File on page 196
 - ▶ Out-of-the-Box Converters on page 196
 - ▶ Plug-ins on page 200
 - ▶ Configuration Examples on page 201
 - ▶ Federated Database Log Files on page 211
 - ▶ External References on page 214
- Troubleshooting and Limitations** on page 214

Concepts

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.

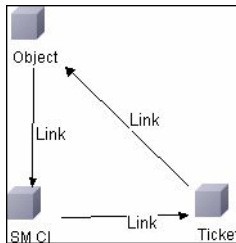
This version of the database adapter implementation is based on a JPA (Java Persistence API) standard with the Hibernate ORM library as a persistence provider.

Non-supported TQL Queries

The following limitations exist on the federated CMDB only:

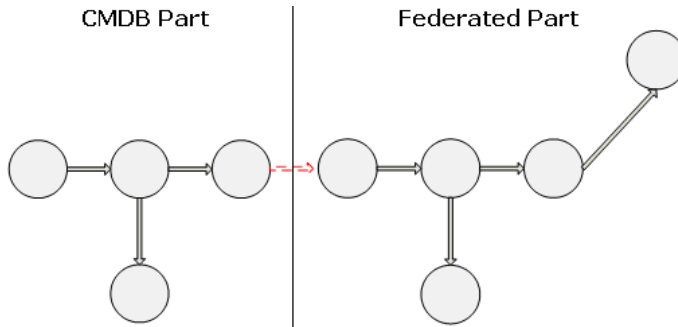
- subgraphs are not supported
- compound relationships are not supported
- cycles or cycle parts are not supported

The following TQL is an example of a cycle:



- Function layout is not supported.
- 0..0 cardinality is not supported.
- The Joinf 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 CMDB side is mapped to a federated entity called reconciliation CIT.

Mapping. Each attribute in the UCMDB 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 CMDB in one format and in the federated database in another format).

Note:

- To connect the UCMDB and external database source, an appropriate association must exist in the database. For details, see "Prerequisites" on page 158.
 - Reconciliation with CMDB 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 214.

In an elementary mapping, each Java class is mapped to a single table. More advanced mapping enables inheritance mapping (as can occur in the CMDB 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 page 157.

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

This section also includes the following topics:

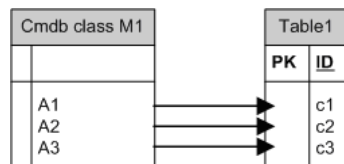
- "Examples of Object-Relational Mapping" on page 155
- "Associations" on page 157
- "Usability" on page 157

Examples of Object-Relational Mapping

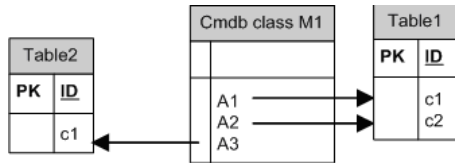
The following examples describe object-relational mapping:

Example of 1 CMDB Class Mapped to 1 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.

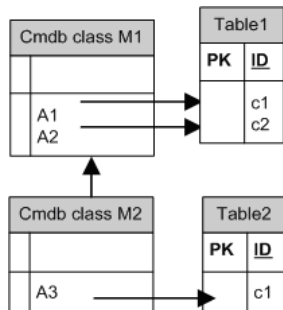


Example of 1 CMDB Class Mapped to 2 Database Tables:



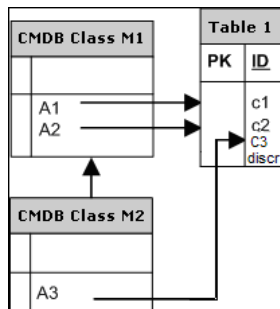
Example of Inheritance:

This case is used in the CMDB, 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.



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 CMDB class. In addition, there is only one possible relation type between the federated class and the non-federated class.

Tasks

Validate Adapter Setup

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

This task includes the following steps:

- "Prerequisites" on page 158
- "Create a CI Type" on page 160
- "Create a Relationship" on page 161

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 CMDB IDs that matches the CMDB IDs of the nodes in the CMDB. For details on reconciliation, see "Reconciliation" on page 154.

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 155.

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

<code>NODE_ID</code>	<code>CARD_ID</code>	<code>CARD_TYPE</code>	<code>CARD_NAME</code>
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

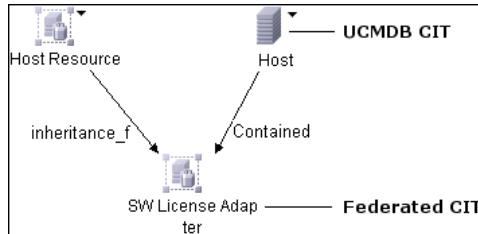
In this step you create a federated CIT that is to be mapped to data in the RDBMS (the external data source).

- a** In UCMDB, access the CI Type Manager and create a new CI Type. For details, see "Create a CI Type" in the *Modeling Guide*.
- b** 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 CMDB views.

3 Create a Relationship

In this step you add a relationship between the UCMDB CIT and the new CIT that represents the data to be federated 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*.

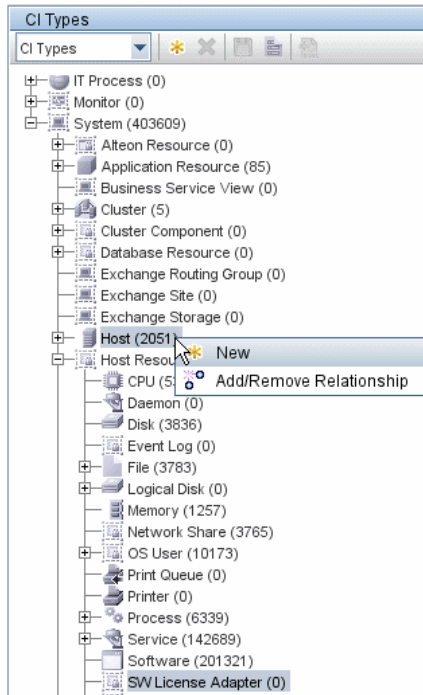


Note: At this stage, you cannot yet view the federated data, as you have not yet defined the method for bringing in the data.

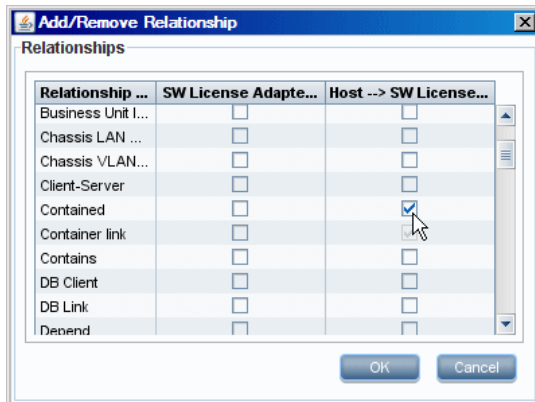
Example of Creating a Containment Relationship:

Prepare the Adapter Package

1 In the CIT Manager, select the two CITs:



2 Create a **Containment** relationship between the two CITs:



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

- 1 Locate the **dbAdapter.zip** package in the **C:\hp\UCMDB\UCMDBServer\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:

```
<adapter 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 replication data, 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 Points pane in HP Universal CMDB.

For details about populating the CMDB with data, see "Integration Studio Page" in the *HP Universal CMDB Data Flow Management Guide*.

- 4 In the temporary directory, open the **adapterCode** folder and rename **GenericDBAdapter** to the value of **adapter id** that was used in step 3.

This folder contains the jar files that execute the federation logic, for example, the adapter name, the queries and classes in the CMDB, and the fields in the RDBMS that the adapter supports.
- 5 Configure the adapter as required. For details, see "Configure the Adapter" on page 164.
- 6 Create a *.zip file with the same name as you gave to the **adapter id** attribute, as described in step %o on page 163.

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: **C:\hp\UCMDB\UCMDBServer\content\adapters**).

Configure the Adapter

You can use either a minimal or advanced method of configuring the adapter.

These configuration files are located in the **dbAdapter.zip** package in the **C:\hp\UCMDB\UCMDBServer\content\adapters** folder that you extracted in step 2 of "Prepare the Adapter Package" on page 162.

Adapter Configuration – Minimal Method

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.

The following procedure describes a method of mapping the class model in the UCMDB to an RDBMS. You would use this minimal method when you need to:

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

This method:

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

Configure the adapter.conf File

In this step, you change the settings in the `adapter.conf` file so that the data is federated automatically.

- 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 UCMDB to the fields in the RDBMS table.

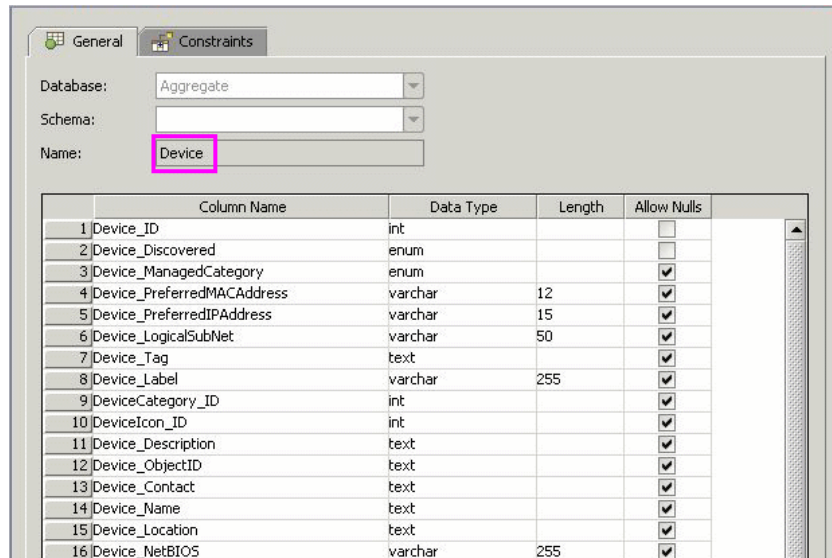
- 1** Open the `simplifiedConfiguration.xml` file in a text editor.
This file includes a template that you use for each entity to be mapped.
- 2** Make changes to the following attributes:
 - The CIT name in Universal CMDB (`cmdb-class-name`) and the corresponding table name in the RDBMS (`default-table-name`):

```
<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:



- 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 Universal CMDB (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 CMDB 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 Universal CMDB 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.

Adapter Configuration – Advanced Method

Configure the `orm.xml` File

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

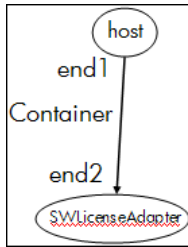
- 1 Open the `orm.xml` file in a text editor.

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

- 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:

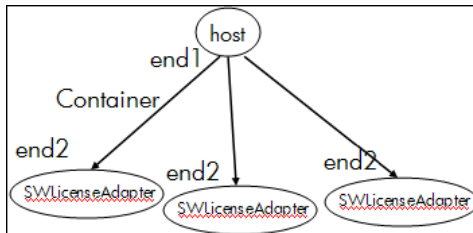
► One to one:



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>
```

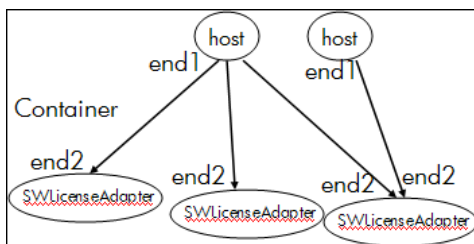
► Many to one:



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 188.

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 Universal CMDB 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 Universal CMDB 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 160:

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

- ▶ The name of the corresponding table in the RDBMS:

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

- ▶ The two primary key nodes:

```
<id class="generic_db_adapter.IDClass2PK_SW_Adapter">
```

- ▶ The unique identity 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 attribute name in the Universal CMDB CIT and the name of the corresponding attribute in the RDBMS:

```
<basic name="license_required">  
  <column updatable="false" insertable="false"  
  name="MyAdapter_LicenseRequired"/>
```

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

- The class of the Universal CMDB 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 Universal CMDB 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

- 1 Open the `reconciliation_types.txt` file in a text editor.

For details, see "The reconciliation_types.txt file" on page 189.

Configure the reconciliation_rules.txt File

In this step you define the rules by which the adapter reconciles the Universal CMDB and the RDBMS.

- 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 189.

Deploy the Adapter

- 1 In UCMDDB, access the Package Manager. For details, see "Package Manager Page" in the *HP UCMDDB 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 XML file contents is recognized by Package Manager.

Edit the Adapter

Once you have created and deployed the adapter, you can then edit it within UCMDB. For details, see "Adapter Management" on page 125.

Load the Adapter

In this step you load the adapter onto the UCMDB machine.

Note: Every time you make a change to the adapter, you must redeploy it using the JMX console.

- 1 On the HP Universal CMDB server machine, launch the Web browser and enter the following address:

```
http://<machine name or IP address>.<domain_name>:8080/jmx-console
```

where **<machine name or IP address>** is the machine on which UCMDB is installed. You may have to log in with the user name and password.

- 2 Click the **service=Fcmdb Config Services** link under the Topaz section.
- 3 In the JMX MBEAN View page, locate the **loadOrReloadCodeBaseForAdaptorId()** operation.
- 4 In the customerID field, enter **1**.
- 5 In the adaptorId field, enter **MyAdapter**. (This is the name you gave to the adapter.)
- 6 Click **Invoke**.

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 UCMDB, access the Integration Studio (**Data Flow Management > Integration Studio**).
- 2** Create an integration point. For details, see "Create New Integration Point/Edit Integration Properties Dialog Box" in the *HP Universal CMDB Data Flow Management Guide*.

The Federation tab displays all CITs that support federation. For details, see "Federation Tab" in the *HP Universal CMDB 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 UCMDB access the Modeling Studio (**Modeling > Modeling Studio**).
- 2** Create a view. For details, see "Create a Template Based View" in the *Modeling Guide*.

- 3 You can add conditions to the TQL, for example, the last access time is greater than six months:

The screenshot displays the SWLicenseAdapter application window. The main area shows a hierarchical diagram with a 'Host' icon at the top, connected by a 'Container link' to an 'SW License Adapter' icon below it. The 'Information Pane' at the bottom contains a query editor with the following text:

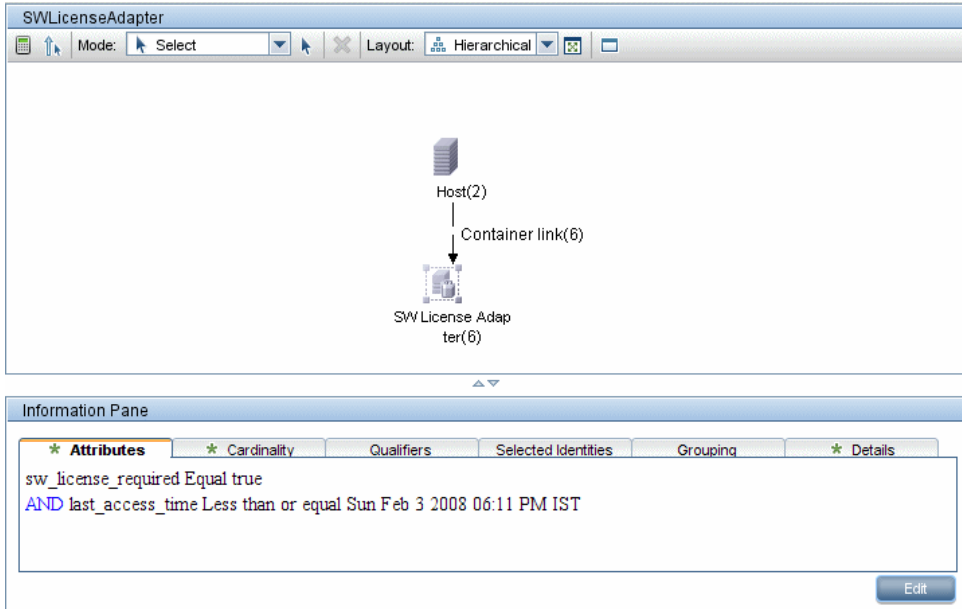
```
sw_license_required Equal true
AND last_access_time Less than or equal Sun Feb 3 2008 06:11 PM IST
```

An 'Edit' button is located at the bottom right of the information pane.

Calculate the Results

In this step you check the results.

- 1 In UCMDB access the Modeling Studio (**Modeling > Modeling Studio**).
- 2 Calculate results by clicking the Calculate TQL result count button.



The screenshot displays the UCMDB Modeling Studio interface. The top window, titled "SWLicenseAdapter", shows a hierarchical diagram with a "Host(2)" node connected to a "SW License Adapter(6)" node via a "Container link(6)". The "Information Pane" below shows the following details:

* Attributes	* Cardinality	Qualifiers	Selected Identities	Grouping	* Details
sw_license_required	Equal	true			
AND last_access_time	Less than or equal	Sun Feb 3 2008 06:11 PM IST			

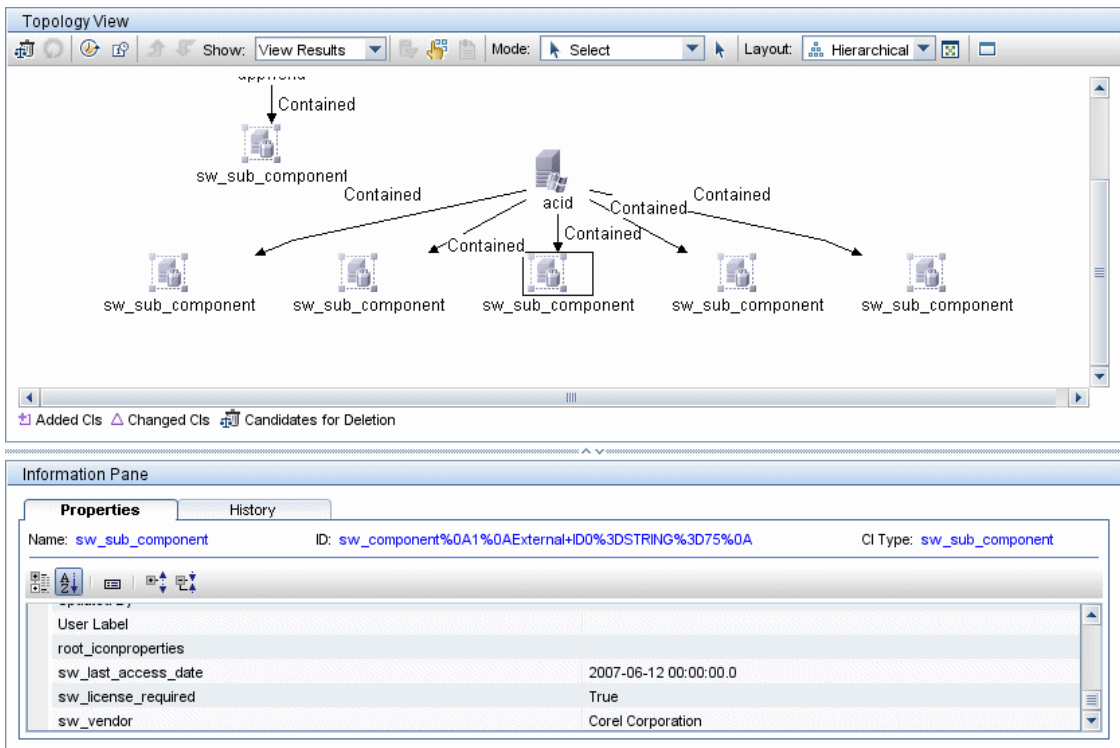
An "Edit" button is located at the bottom right of the information pane.

- 3 Click the **Preview** button to view the CIs in the CIT.

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 UCMDB access the IT Universe Manager (**Modeling > IT Universe Manager**).
- 2 The Properties tab displays the results of the federation:



The screenshot displays the UCMDB IT Universe Manager interface. The top window, titled "Topology View", shows a hierarchical diagram of components. At the top is a component labeled "sw_sub_component". Below it, another "sw_sub_component" is shown, which is contained by the one above. This second "sw_sub_component" is further divided into five sub-components, all labeled "sw_sub_component". The central component in this group is labeled "acid". Arrows labeled "Contained" indicate the relationships between the components.

The bottom window, titled "Information Pane", shows the "Properties" tab for a selected component. The component name is "sw_sub_component", and its CI Type is "sw_sub_component". The ID is "sw_component%0A1%0AEExternal+ID0%3DSTRING%3D75%0A". The properties are as follows:

Property Name	Value
User Label	
root_jconproperties	
sw_last_access_date	2007-06-12 00:00:00.0
sw_license_required	True
sw_vendor	Corel Corporation

View Reports

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

sw_component_report - Required License Software		Not used in 6 months null	
Set View-specific Report (Optional): [Clear]			
..			
[-] sw_sub_component (sw_sub_component) (Path: appworld (Host))			
Sw Name:	Photoshop	sw_license_required:	True
sw_last_access_date:		sw_vendor:	Adobe
[-] acid (Host)			
Host Name: acid			
[-] sw_sub_component (sw_sub_component) (Path: acid (Host))			
Sw Name:	MS Word	sw_license_required:	True
sw_last_access_date:		sw_vendor:	Microsoft
[-] sw_sub_component (sw_sub_component) (Path: acid (Host))			
Sw Name:	PcAnywhere	sw_license_required:	True
sw_last_access_date:	Sun Apr 8, 2007 12:00 AM	sw_vendor:	Symantec
[-] sw_sub_component (sw_sub_component) (Path: acid (Host))			
Sw Name:	TextPad 4	sw_license_required:	True
sw_last_access_date:	Fri May 11, 2007 12:00 AM	sw_vendor:	Helios Software Solutions
[-] sw_sub_component (sw_sub_component) (Path: acid (Host))			
Sw Name:	WinZip	sw_license_required:	True

Enable Log Files

To understand the calculation flows, adapter lifecycle, and to view debug information, you can consult the log files. For details, see "Federated Database Log Files" on page 211.

Reference

Federated Database Configuration Files

The files discussed in this section are located in the **dbAdapter.zip** package in the **C:\hp\UCMDB\UCMDBServer\content\adapters** folder.

This section includes the following topics:

- "General Configuration" on page 179
- "Advanced Configuration" on page 179
- "Hibernate Configuration" on page 180
- "Simple Configuration" on page 180

General Configuration

- **adapter.conf**. The adapter configuration file. For details, see "The adapter.conf File" on page 180.

Advanced Configuration

- **orm.xml**. The object-relational mapping file in which you map between CMDB CITs and database tables. For details, see "The orm.xml File" on page 183.
- **reconciliation_rules.txt**. Contains the reconciliation rules. For details, see "The reconciliation_rules.txt File (for backwards compatibility)" on page 189.

- ▶ **transformations.txt.** Transformations file in which you specify the converters to apply to convert from the CMDB value to the database value, and vice versa. For details, see "The transformations.txt File" on page 192.

Hibernate Configuration

- ▶ **persistence.xml.** Used to override out of the box Hibernate configurations. For details, see "The persistence.xml File" on page 193.

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 181.



The adapter.conf File

This file contains the following settings:

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

Note: Usage of 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.

The simplifiedConfiguration.xml File

This file is used for simple mapping of CMDB classes to database tables. The template for editing the file is located in the **C:\hp\UCMDBServer\j2f\fcmdb\CodeBase\GenericDBAdapter\META-INF** directory.

This section includes the following topics:

- "The simplifiedConfiguration.xml File Template" on page 181
- "Limitations" on page 183

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 Universal CMDB 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.fcmdb.adapters.dbAdapter.dal.transform.FcmdbDalTransformerFromExternalDB`
- ▶ `com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.FcmdbDalTransformerToExternalDB`

Use these converters if the value in the CMDB and in the database are not the same. For example, the node name in the CMDB has the suffix `mer.com`.

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-
converter="com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.GenericE
numTransformer(generic-enum-transformer-example.xml)" to-CMDB-
converter="com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.GenericE
numTransformer(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 one node TQLs only (in the database source). For example, you can run a `node > ticket` and a `ticket` TQL. 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 CMDB 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 CMDB CITs to database tables.

A template to use for creating a new file is located in the **C:\hp\UCMDBServer\j2f\fcmdb\CodeBase\GenericDBAdapter\META-INF** directory.

This section includes the following topics:

- "The orm.xml File Template" on page 184
- "Multiple ORM files" on page 188
- "Naming Conventions" on page 188
- "Using Inline SQL Statements Instead of Table Names" on page 188

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 Universal CMDB 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 CMDB 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>
  </attributes>
</entity>
```

The following example shows a CMDB 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>
```

Multiple ORM files

Multiple mapping files are supported from version 7.5.1. 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`.
- ▶ CMDB CITs, which are also reserved words in Java, should be prefixed by **gdba_**. For example, for the CMDB 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 reconciliation_types.txt file

This file is used to configure the reconciliation types.

Each row in the file represents a CMDB CIT that is connected to a federated database CIT in the TQL

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 UCMDB 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 CMDB CIT that is connected to the federated database CIT in the TQL).

This expression includes the logic that decides whether two multinodes are equal (one multinode in the UCMDB 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 multinodes are 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 (^) sign.

The parameters end1_type, end2_type and link_type are used only if the reconciliation TQL contains two nodes and not just a multinode. In this case, the reconciliation TQL 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 UCMDB 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 UCMDB 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 UCMDB 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 should have no more than two nodes. For example, the **node > ticket** TQL has a node in the UCMDB 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 CMDB is in the format of *.m.com, a converter can be used from CMDB 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 two entities (the defined association can also be made in a node table):

DB Host	
PK	<u>host_id</u>
	ex_host_name ex_ip_address

DB Ticket	
PK	<u>ticket_id</u>
	host_id

Note: Both tables must be part of the federated RDBMS source and not the CMDB 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.fcldb.adapters.dbAdapter.dal.
transform.impl.GenericEnumTransformer(generic-enum-transformer-example.xml)]
from_DB_class[com.mercury.topaz.fcldb.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.fcldb.adapters.dbAdapter.dal.transform.FcldbDalTransformerToExternalDB. The elements in the parenthesis are given to this class constructor. Use this converter to transform CMDB 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.fcldb.adapters.dbAdapter.dal.transform.FcldbDalTransformerFromExternalDB** interface. The elements in the parenthesis are given to this class constructor. Use this converter to transform database values to CMDB values, for example, to append the suffix of **.com** to each node name.

For details, see "Out-of-the-Box Converters" on page 196.

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 MSSQL 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.

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>
```

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.

Example of Discriminator Mapping:

The discriminator.properties file includes the following code:

```
node=10001, 10005,10010,10011,10012
nt=10002,10003
unix=10004,10006,10008
```

The orm.xml file includes the following code:

```
<entity class="generic_db_adapter.node" >
  <table name="[table_name]"/>
  ...
  <inheritance strategy="SINGLE_TABLE"/>
  <discriminator-value>node</discriminator-value>
  <discriminator-column name="[discriminator_column]"/>
  ...
</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 calculated as follows:

- ▶ The discriminator column of the corresponding table contains 10002 for a certain entry. The entry is mapped to the **nt** CIT.
- ▶ The discriminator column of the corresponding table contains 10006 for a certain entry. The entry is mapped to the **unix** CIT.
- ▶ The discriminator column of the corresponding table contains 10010 for a certain entry. The entry is mapped to the **node** CIT.

Note that 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 plug-in. For details, see "Plug-ins" on page 200.

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]
```

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:

- "The enum-transformer Converter" on page 197
- "The SuffixTransformer Converter" on page 199
- "The PrefixTransformer Converter" on page 199
- "The BytesToStringTransformer Converter" on page 200

The enum-transformer Converter

This converter uses an XML file that is given as an input parameter.

The XML file maps between hard-coded CMDB 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.

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.

Example of the 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: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:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```

        <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: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:complexType>
</xs:element>
</xs:schema>

```

Example of Converting 'sys' Value to 'System' Value:

In this example, sys value in the CMDB is transformed into System value in the federated database, and System value in the federated database is transformed into sys value in the CMDB.

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 CMDB or federated database source value.

There are two implementations:

- **com.mercury.topaz.fcmbd.adapters.dbAdapter.dal.transform.impl.AdapterToCmdbAddSuffixTransformer.** Adds the suffix (given as input) when converting from federated database value to CMDB value and removes the suffix when converting from CMDB value to federated database value.
- **com.mercury.topaz.fcmbd.adapters.dbAdapter.dal.transform.impl.AdapterToCmdbRemoveSuffixTransformer.** Removes the suffix (given as input) when converting from federated database value to CMDB value and adds the suffix when converting from CMDB value to federated database value.

The PrefixTransformer Converter

This converter is used to add or remove a prefix from the CMDB or federated database value.

There are two implementations:

- **com.mercury.topaz.fcmbd.adapters.dbAdapter.dal.transform.impl.AdapterToCmdbAddPrefixTransformer.** Adds the prefix (given as input) when converting from federated database value to CMDB value and removes the prefix when converting from CMDB value to federated database value.
- **com.mercury.topaz.fcmbd.adapters.dbAdapter.dal.transform.impl.AdapterToCmdbRemovePrefixTransformer.** Removes the prefix (given as input) when converting from federated database value to CMDB value and adds the prefix when converting from CMDB value to federated database value.

The BytesToStringTransformer Converter

This converter is used to convert byte arrays in the UCMDB to their string representation in the federated database source.

The converter is:

com.mercury.topaz.fcldb.adapters.dbAdapter.dal.transform.impl.CmdbToAdapterBytesToStringTransformer.

Plug-ins

The generic database adapter supports the following plug-ins:

- ▶ An optional plug-in for full topology synchronization.
- ▶ A mandatory plug-in for synchronizing changes in topology.
- ▶ An optional plug-in for synchronizing layout.
- ▶ An optional plug-in to retrieve supported queries for synchronization. If this plug-in is not defined, all TQL names are returned.
- ▶ An internal, optional plug-in to change the TQL definition and TQL result.
- ▶ An internal, optional plug-in to change a layout request and CIs result.
- ▶ An internal, optional plug-in to change a layout request and relationships result.

The plug-ins are configured using the **plugins.txt** file under the META-INF folder of the adapter.

Implementing the mandatory plug-in for synchronizing changes in topology

To implement the plugin, write a Java class implementing **com.mercury.topaz.fcldb.adapters.dbAdapter.plugin.FcldbPluginForSyncGetChangesTopology**, which has a single method. For reference, use **com.hp.ucldb.adapters.ed.plugins.replication.EDReplicationPlugin**.

To view the source of this class, do the following:

- 1** Locate the **DDMiAdapter.zip** package in the **C:\hp\UCMDB\UCMDBServer\content\adapters** folder.
- 2** Extract **adapterCode/DDMiAdapter/DDMiAdapter.jar**.
- 3** Decompile **com\hp\ucmdb\adapters\ed\plugins\replication\EDReplicationPlugin.class**.
- 4** Update the **plugins.txt** file with the fully qualified name of your class.
- 5** Pack your class into a jar file and put it under the **adapterCode\<adapter id>** folder in the adapter package, prior to deploying it.

Configuration Examples

This section gives examples of configurations.

This section includes the following topics:

- "Use Case" on page 201
- "Single Node Reconciliation" on page 202
- "Two Node Reconciliation" on page 204
- "Using a Primary Key that Contains More Than One Column" on page 208
- "Using Transformations" on page 210

Use Case

Use case. A TQL is:

```
node > (composition) > card
```

where:

node is the UCMDB 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 183.

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"/>
        <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>
    </attributes>
  </entity>

```

```
</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 189.

```
node
```

The reconciliation_rules.txt File

For details, see "The reconciliation_rules.txt File (for backwards compatibility)" on page 189.

```
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 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>
```

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 189.

```
node
```

The reconciliation_rules.txt File

For details, see "The reconciliation_rules.txt File (for backwards compatibility)" on page 189.

```
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 183.

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" updatable="false"/>
      <generated-value strategy="TABLE"/>
    </id>
    <id name="id2">
      <column name="hwBusesSupported_Seq" insertable="false" updatable="false"/>
      <generated-value strategy="TABLE"/>
    </id>
    <id name="id3">
      <column name="hwCards_Seq" insertable="false" updatable="false"/>
      <generated-value strategy="TABLE"/>
    </id>
  .
  .
  .
<entity class="generic_db_adapter.node_containment_card" >
  <table name="hwCards"/>
  <attributes>
    <id name="id1">
      <column name="Device_ID" insertable="false" updatable="false"/>
      <generated-value strategy="TABLE"/>
    </id>
    <id name="id2">
      <column name="hwBusesSupported_Seq" insertable="false" updatable="false"/>
      <generated-value strategy="TABLE"/>
    </id>
    <id name="id3">
      <column name="hwCards_Seq" insertable="false" updatable="false"/>
    </id>
```

```

        <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="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.fcldb.adapters.dbAdapter.dal.transform.impl.GenericE
numTransformer(generic-enum-transformer-example.xml)" to-CMDB-
converter="com.mercury.topaz.fcldb.adapters.dbAdapter.dal.transform.impl.GenericE
numTransformer(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.fcldb.adapters.dbAdapter.dal.transform.impl.Generic
EnumTransformer(generic-enum-transformer-example.xml)]
from_DB_class[com.mercury.topaz.fcldb.adapters.dbAdapter.dal.transform.impl.Gene
ricEnumTransformer(generic-enum-transformer-example.xml)]
```

Federated Database 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" on page 212
- "Log Locations" on page 212

Log Levels

You can configure the log level for each of the logs.

In a text editor, open the **C:\hp\j2f\conf\core\Tools\log4j\fcmdb\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.fcmbd.gdba.dal.SQL=${loglevel},fcmbd.gdba.dal.SQL.appender
```

to:

```
log4j.category.fcmbd.gdba.dal.SQL=INFO,fcmbd.gdba.dal.SQL.appender
```

Log Locations

The log files are located in the **C:\hp\j2f\log\fcmbd** directory.

➤ Fcmbd.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).

➤ fcmbd.log

Consult for exceptions.

► **cmdb.log**

Consult for exceptions.

► **Fcmdb.gdba.mapping.engine.log**

The mapping engine log. Gives details about the reconciliation TQL that the mapping engine uses, and the reconciliation topologies that are compared during the connect phase.

Consult this log when a TQL 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 TQLs and their results.

Consult this log when a TQL 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 155.

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.

JPA Limitations

- ▶ All tables must have a primary key column.
- ▶ CMDB 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 CMDB 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 inherit from a multinode (in most cases this means a new federated CIT cannot be created underneath the node CIT)
- ▶ Federated CITs cannot inherit from another federated CIT, unless they are both located in the same data store.

- To view federated data, you must add the federated CIT itself to a TQL, and not its parent CIT. (If you use the parent CIT, the federated instances will not appear in the results.)
- Federated CITs cannot be trigger CITs in a correlation rule but they can be included in a correlation TQL.
- 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).
- Properties from the CI Type list are not supported.
- Using a class qualifier in a condition is not supported.
- Subgraphs are not supported.
- Compound relationships are not supported.
- The external CI CMDB id 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.

8

HP ServiceCenter/Service Manager Adapter

This chapter includes:

Concepts

- ▶ Adapter Usage on page 218
- ▶ The Adapter Configuration File on page 220
- ▶ Multi-Threading on page 228

Tasks

- ▶ Deploy the Adapter – Typical Deployment on page 228
- ▶ Deploy the ServiceDesk Adapter on page 229
- ▶ Add an Attribute to the ServiceCenter/Service Manager CIT on page 233
- ▶ Communicate with Service Manager over SSL on page 241

Note: This Adapter is a specific configuration of the ServiceDesk Adapter.

Concepts

Adapter Usage

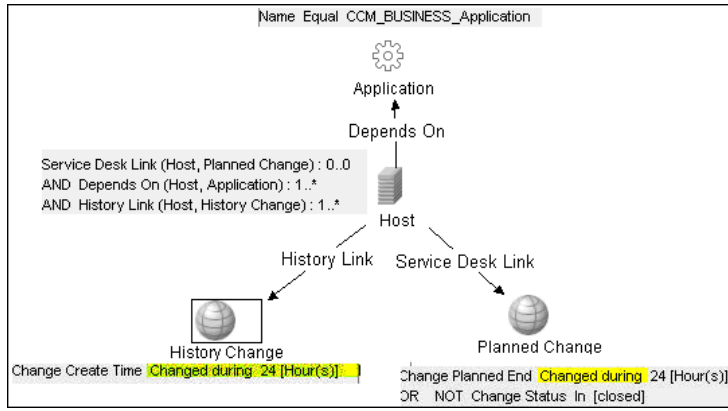
The ServiceCenter/Service Manager Adapter supports the retrieval of data from HP ServiceCenter and HP Service Manager. This adapter connects to, and receives data from, ServiceCenter/Service Manager using the Web Service API. Every request to ServiceCenter/Service Manager to calculate a federated query is made through this adapter. The Adapter is compatible with HP Universal CMDB, version 8.0 or later, HP ServiceCenter, version 6.2, and HP Service Manager, versions 7.0x, 7.1x (following changes to the WSDL configuration).

The Adapter supports three external CI types: Incident, Problem, and Planned Change. The adapter retrieves the CIs of these types from ServiceCenter/Service Manager with the required layout and by a given filter (using reconciliation and/or a CI filter). Each of these CITs can be related to one of the following UCMDB internal CITs: Host, Business Service, Application. Each UCMDB internal CIT includes a reconciliation rule in the ServiceCenter/Service Manager configuration that can be changed dynamically (for details, see "Reconciliation Data Configuration" on page 223). Note that there are no internal relationships between Adapter-supported CITs.

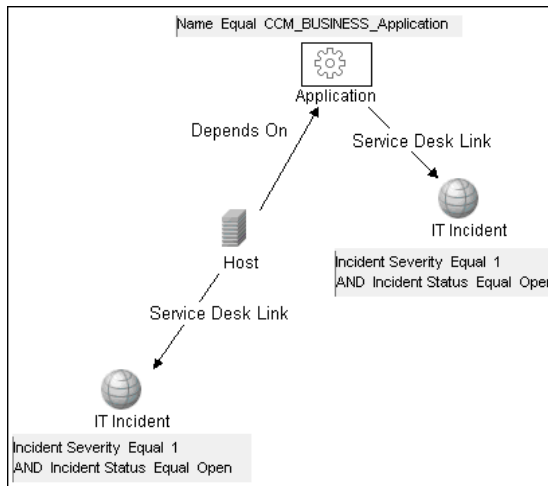
The modeling of the supported CITs and virtual relationships is supplied with the Adapter. You can add attributes to a CIT (for details, see "Add an Attribute to the ServiceCenter/Service Manager CIT" on page 233).

The following use cases (that include TQL examples) describe how the Adapter can be employed:

- 1 A user needs to display all unplanned changes to all hosts running a specific application during the last 24 hours:



- 2 A user needs to see all open critical incidents on an application and its hosts:



The Adapter Configuration File

The `serviceDeskConfiguration.xml` Adapter configuration file contains three parts:

- 3** The first part, which is defined by the `ucmdbClassConfigurations` element, contains the external CIT configuration that the Adapter supports. For details, see "External CITs Configuration" on page 220.
- 4** The second part, defined by the `reconciliationClassConfigurations` element, contains reconciliation data information for appropriate UCMDDB CITs. For details, see "Reconciliation Data Configuration" on page 223.
- 5** The third part, defined by the `globalConnectorConfig` element, includes the global configuration for a specific connector implementation. For details, see "Global Configuration" on page 227.

This section also includes the following topics:

- ▶ "External CITs Configuration" on page 220
- ▶ "Reconciliation Data Configuration" on page 223
- ▶ "Global Configuration" on page 227

External CITs Configuration

Each CIT that is supported by the Adapter is defined in the first section of the Adapter configuration file.

This section, `ucmdbClassConfiguration`, represents the only supported CIT configuration. This element contains the CIT name as defined in the UCMDDB class model (the `ucmdbClassName` attribute), mapping for all its attributes (the `attributeMappings` element), and a private configuration for a specific connector implementation (the `classConnectorConfiguration` element):

- ▶ The `ucmdbClassName` attribute defines the UCMDDB class model name.
- ▶ The `attributeMappings` element contains `attributeMapping` elements.

The `attributeMapping` element defines the mapping between the UCMDB model attribute name (the `ucmdbAttributeName` attribute) to an appropriate ServiceCenter/Service Manager attribute name (the `serviceDeskAttributeName` attribute).

For example:

```
<attributeMapping ucmdbAttributeName="problem_brief_description"
serviceDeskAttributeName="brief.description"/>
```

This element can optionally contain the following converter attributes:

- ▶ The `converterClassName` attribute. This is the converter class name that converts the UCMDB attribute value to the ServiceDesk attribute value.
- ▶ The `reversedConverterClassName` attribute. This is the converter class name that converts the ServiceDesk attribute value to the UCMDB attribute value.
- ▶ The `classConnectorConfiguration` element contains the configuration for the specific connector implementation for the current external CIT. Wrap this configuration in CDATA if it contains special XML characters (for example, `&`; replacing `&`).

The useful fields of the Service Manager `classConnectorConfiguration` element are as follows:

- ▶ The `device_key_property_names` element contains the fields names in the WSDL information of the current object that can contain the device ID (for example, `ConfigurationItem`). Each field should be added as a `device_key_property_name` element.
- ▶ The `id_property_name` element contains the field name in the WSDL information that contains the ID of the current object.

The following example shows the ucmdbClassConfiguration section of the serviceDeskConfiguration.xml file. The section includes the ucmdbClassName element for the Incident CIT with a ServiceCenter connector implementation:

```
<ucmdbClassConfiguration ucmdbClassName="it_incident">
  <attributeMappings>
    <attributeMapping ucmdbAttributeName="incident_id"
serviceDeskAttributeName="IncidentID"/>
    <attributeMapping ucmdbAttributeName="incident_brief_description"
serviceDeskAttributeName="BriefDescription"/>
    <attributeMapping ucmdbAttributeName="incident_category"
serviceDeskAttributeName="Category"/>
    <attributeMapping ucmdbAttributeName="incident_severity"
serviceDeskAttributeName="severity"/>
    <attributeMapping ucmdbAttributeName="incident_open_time"
serviceDeskAttributeName="OpenTime"/>
    <attributeMapping ucmdbAttributeName="incident_update_time"
serviceDeskAttributeName="UpdatedTime"/>
    <attributeMapping ucmdbAttributeName="incident_close_time"
serviceDeskAttributeName="ClosedTime"/>
    <attributeMapping ucmdbAttributeName="incident_status"
serviceDeskAttributeName="IMTicketStatus"/>
  </attributeMappings>

  <classConnectorConfiguration>
    <![CDATA[ <class_configuration
connector_class_name="com.mercury.topaz.fcmdb.adapters.serviceDeskAdapter.servi
ceCenterConnector.impl.SimpleServiceCenterObjectConnector">
      <device_key_property_names>
        <device_key_property_name>ConfigurationItem</device_key_property_name>
      </device_key_property_names>
      <id_property_name>IncidentID</id_property_name>
      <keys_action_info>
        <request_name>RetrieveUcmdbIncidentKeysListRequest</request_name>
      <response_name>RetrieveUcmdbIncidentKeysListResponse</response_name>
      </keys_action_info>
      <properties_action_info>
        <request_name>RetrieveUcmdbIncidentListRequest</request_name>
        <response_name>RetrieveUcmdbIncidentListResponse</response_name>
      </properties_action_info>
    </class_configuration> ]]>
  </classConnectorConfiguration>
</ucmdbClassConfiguration>
```

Adding an Attribute to a CIT

When adding an attribute to the UCMDB model for an Adapter-supported CIT:

- 1 Navigate to **Data Flow Management > Adapter Management >** and select the **ServiceManagerAdapter** that corresponds to your version of Service Manager.
- 2 Select **Configuration Files > ServiceDeskConfiguration.xml** file and add an **attributeMapping** element to the appropriate **ucmdbClassConfiguration** element.
- 3 Verify that ServiceCenter/Service Manager externalizes this attribute in its Web Service API.
- 4 Click **Save**.

Reconciliation Data Configuration

Each UCMDB CIT that can be related to the adapter-supported CIT is defined in the second section of the Adapter configuration file.

This section, **reconciliationClassConfigurations**, represents the reconciliation data configuration for one UCMDB CIT. The element includes two attributes:

- ▶ The **ucmdbClassName** attribute. This is the CIT name as defined in the UCMDB class model.
- ▶ The **concreteMappingImplementationClass** attribute. This is the class name of the concrete implementation for the **ConcreteMappingEngine** interface. Use this attribute to map between instances of UCMDB CITs and external Adapter CITs. The default implementation that is used is:

```
com.mercury.topaz.fcmb.adapters.serviceDeskAdapter.mapping.impl.OneNodeMappingEngine
```

An additional implementation exists that is used only for the host reconciliation CIT for reconciliation by the IP of the host:

```
com.mercury.topaz.fcmb.adapters.serviceDeskAdapter.mapping.impl.HostIpMappingEngine
```

The reconciliationClassConfiguration element can contain one of the following elements:

- The reconciliationById element. This element is used when the reconciliation is done by ID. In this case, the text value of this element is the ServiceDesk field name that contains the CMDB ID. For example:

```
<reconciliationById>UcmdbID</reconciliationById>
```

In this example, the ServiceDesk field UcmdbID contains the CMDB ID of the appropriate host.

- The reconciliationData element. Use this element if the reconciliation is done by comparing attributes. You can run reconciliation with one attribute or several attributes by using the logical operators OR and/or AND.

If you run reconciliation with one attribute, the reconciliationData child element should be a reconciliationAttribute element. The reconciliationAttribute element contains an appropriate UCMDb attribute name (the ucmdbAttributeName attribute) and an appropriate ServiceDesk attribute name (the serviceDeskAttributeName attribute). This element can also contain a ucmdbClassName attribute that defines the appropriate UCMDb CIT name. By default, the current reconciliation UCMDb CIT name is used.

You can also use the converterClassName and reversedConverterClassName attributes; they should contain the converter class name that converts the UCMDb attribute value to the ServiceDesk attribute value, or vice versa.

For example:

```
<reconciliationData>  
  <reconciliationAttribute ucmdbAttributeName="name"  
  serviceDeskAttributeName="NetworkName"  
  converterClassName="com.mercury.topaz.fcmdb.adapters.serviceDeskAdapter.converter.PropertyValueConverterToUpperCase"/>  
</reconciliationData>
```

For reconciliation to run with two or more attributes, use a logical operator between reconciliation attributes.

The logical operator AND can contain several reconciliationAttribute elements (the minimum is 2). In this case the reconciliation rule contains an AND operator between attribute comparisons.

For example:

```
<reconciliationData>
<AND>
  <reconciliationAttribute ucmdbAttributeName="name"
serviceDeskAttributeName="NetworkName"
converterClassName="com.mercury.topaz.fcmdb.adapters.serviceDeskAdapter.converter.PropertyValueConverterToUpperCase"/>
  <reconciliationAttribute ucmdbClassName="ip_address"
ucmdbAttributeName="name" serviceDeskAttributeName="NetworkAddress" />
</AND>
</reconciliationData>
```

In this example, the reconciliation rule follows this format: node.name= NetworkName and ip_address.name= NetworkAddress.

The logical operator OR can contain several reconciliationAttribute and AND elements. In this case, the reconciliation rule contains an OR operator between attributes and AND expressions. Since XML does not assure the order of elements, you should provide a priority attribute to each sub-element of OR element type. The comparison between OR expressions is calculated by these priorities.

For example:

```
<reconciliationData>
<OR>
  <reconciliationAttribute ucmdbAttributeName="primary_dns_name"
serviceDeskAttributeName="NetworkDNSName" priority="2" />
<AND priority="1" >
  <reconciliationAttribute ucmdbAttributeName="name"
serviceDeskAttributeName="NetworkName"
converterClassName="com.mercury.topaz.fcmdb.adapters.serviceDeskAdapter.converter.PropertyValueConverterToUpperCase"/>
  <reconciliationAttribute ucmdbClassName="ip_address"
ucmdbAttributeName="name" serviceDeskAttributeName="NetworkAddress" />
</AND>
</OR>
</reconciliationData>
```

In this example the reconciliation rule follows this format: (node.primary_dns_name= NetworkDNSName OR (node.name= NetworkName and ip_address.name= NetworkAddress)). Since the AND element takes a priority attribute of value 1, the (node.name= NetworkName and ip_address.name= NetworkAddress) condition is checked first. If the condition is satisfied, the reconciliation is run. If not, the .host_dnsname= NetworkDNSName condition is checked.

The additional sub-element of the reconciliationClassConfiguration element is classConnectorConfiguration. The classConnectorConfiguration element contains the configuration for a specific connector implementation for the current reconciliation CIT. This configuration should be wrapped by CDATA if it contains some special XML characters (for example, & replacing &).

Changing the Reconciliation Rule of a CIT

- 1 In `serviceDeskConfiguration.xml`, update the appropriate reconciliationData element with the new rule.
- 2 Call to the JMX to reload the adapter: **FCmdb Config Services > loadOrReloadCodeBaseForAdapterId**, using the appropriate customer ID and ServiceDeskAdapter adapter ID, or go to the Integration Points pane and reload the adapter from there. For details, see "Integration Points Pane" in the *HP Universal CMDB Data Flow Management Guide*.

Reconciliation of a Host by ip_address or by host_name

To run reconciliation on a host by ip_address or host_name, place the following ReconciliationData element in the Adapter configuration file:

```
<reconciliationData>
  <OR>
    <reconciliationAttribute priority="1" ucmdbClassName="ip"
ucmdbAttributeName="ip_address" serviceDeskAttributeName="NetworkAddress"/>
    <reconciliationAttribute priority="2" ucmdbClassName="host"
ucmdbAttributeName="host_hostname" serviceDeskAttributeName="NetworkName"
converterClassName="com.mercury.topaz.fcmdb.adapters.serviceDeskAdapter.converter.PropertyValueConverterToUpperCase"/>
  </OR>
</reconciliationData>
```

Global Configuration

The third section of the Adapter configuration file contains the global configuration for the specific connector implementation. This configuration, globalConnectorConfig, should be wrapped by CDATA if it contains some special XML characters (for example, & replacing &).

The useful fields of the Service Manager globalConnectorConfig element are as follows:

- 1 The **date_pattern** element contains the date adapter with which the Service Manager works.

The default is MM/dd/yy HH:mm:ss.

If the date adapter is wrong, an FTQL returns wrong date condition results.

- 2 The **time_zone** element defines the time zone of Service Manager. The default is the UCMDB server time zone.

To check the Service Manager date adapter and time zone:

- a Service Manager version 7:** Access **Menu Navigation > System Administration > Base System Configuration > Miscellaneous > System Information Record**. Click the **Date Info** tab.
- b ServiceCenter version 6.1:** Access **Menu Navigation > Utilities > Administration > Information > System Information**. Click the **Date Info** tab.
- 3** The **max_query_length** element defines the maximal query length in a Service Manager Web service request. The default value is 1000000.
- 4** The **name_space_uri** element defines the name space URI to connect to the Service Manager Web service. The default value is `http://servicecenter.peregrine.com/PWS`.
- 5** The **web_service_suffix** element defines the Service Manager Web service center URI suffix. The default value is `sc62server/ws`. It is used when the URL is created.

Multi-Threading

By default, the ServiceDesk Adapter uses six concurrent threads to push data to Service Manager. To configure the ServiceDesk Adapter multi-thread settings, edit the **sm.properties** file, located in:

Data Flow Management > Adapter Management > ServiceManagerAdapter
corresponding to **Service Manager version > Configuration Files**

Tasks

Deploy the Adapter – Typical Deployment

This section describes a typical deployment of the adapter.

This task includes the following steps:

- 1** "Deploy the ServiceDesk Adapter" on page 229

- a "Add a ServiceCenter/Service Manager External Data Source" on page 229
 - b "Configure HP ServiceCenter 6.2" on page 230 (when connecting to HP ServiceCenter)
 - c "Configure HP Service Manager 7.0/7.1" on page 233 (when connecting to HP Service Manager)
- 2** "Add an Attribute to the ServiceCenter/Service Manager CIT" on page 233
- a "Add an Attribute to the UCMDDB Model" on page 234
 - b "Export Attributes from HP ServiceCenter by Changing the Configuration" on page 235 (when connecting to HP ServiceCenter)
 - c "Export Attributes from HP Service Manager by Changing the Configuration" on page 237 (when connecting to HP Service Manager)
 - d "Modify the Adapter Configuration File" on page 240

Deploy the ServiceDesk Adapter

This section explains where to place the files needed for deployment.

This task includes the following steps:

- "Add a ServiceCenter/Service Manager External Data Source" on page 229
- "Configure HP ServiceCenter 6.2" on page 230
- "Configure HP Service Manager 7.0/7.1" on page 233

1 Add a ServiceCenter/Service Manager External Data Source

In this step, you add an integration point.

- a In UCMDDB, select **Data Flow Management > Integration Studio**.
- b Click the **Create New Integration Point** button to add an integration point. Select the **ServiceDeskAdapter** that matches your version of Service Manager and fill in the mandatory fields.

For help with this dialog box, see "Integration Points Pane" in the *HP Universal CMDB Data Flow Management Guide*.

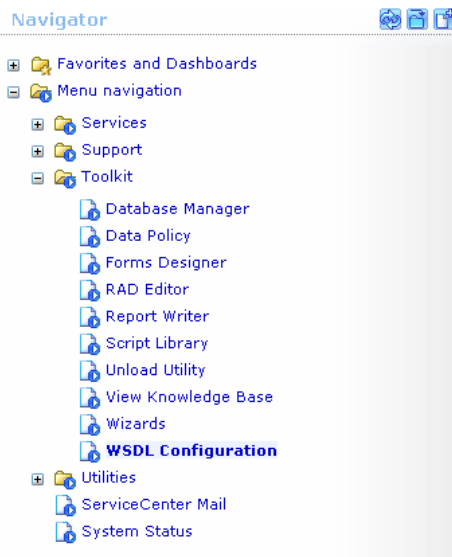


- c Continue with "Configure HP ServiceCenter 6.2" on page 230 or "Configure HP Service Manager 7.0/7.1" on page 233.

2 Configure HP ServiceCenter 6.2

If you are connecting to HP ServiceCenter 6.2, perform the following procedure. If you are connecting to HP Service Manager 7.0/7.1, skip this step.

- a Open HP ServiceCenter, then the ServiceCenter client.
- b Display **WSDL Configuration** in the Navigator (**Main Menu > Menu navigation > Toolkit**):



- c In the Name field, enter **device** and press **Enter**:

Search External Access Definition Records

External Access Definition

Service Name:

Name: Object Name:

Allowed Actions	Action Names
<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>

- d Select the **Data Policy** tab and ensure that the **network.name** attribute is not empty (its value should be **NetworkName**). Change the value to **false**. Save your changes.

Service Name:

Name: Object Name:

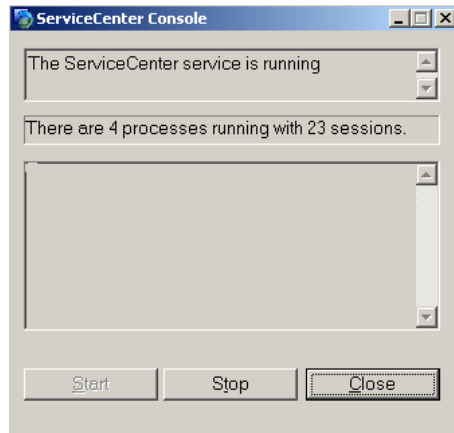
mac.address	<input type="text"/>	<input type="text" value="true"/>	<input type="text"/>	<input type="button" value="v"/>
manufacturer	<input type="text"/>	<input type="text" value="true"/>	<input type="text"/>	<input type="button" value="v"/>
model	<input type="text" value="Model"/>	<input type="text" value="false"/>	<input type="text"/>	<input type="button" value="v"/>
mtbf	<input type="text"/>	<input type="text" value="true"/>	<input type="text"/>	<input type="button" value="v"/>
network.address	<input type="text"/>	<input type="text" value="true"/>	<input type="text"/>	<input type="button" value="v"/>
network.name	<input type="text" value="NetworkName"/>	<input type="text" value="false"/>	<input type="text"/>	<input type="button" value="v"/>
nm.id	<input type="text"/>	<input type="text" value="true"/>	<input type="text"/>	<input type="button" value="v"/>
nondevice	<input type="text"/>	<input type="text" value="true"/>	<input type="text"/>	<input type="button" value="v"/>
objid	<input type="text"/>	<input type="text" value="true"/>	<input type="text"/>	<input type="button" value="v"/>
operating.system	<input type="text"/>	<input type="text" value="true"/>	<input type="text"/>	<input type="button" value="v"/>
order.line.item	<input type="text"/>	<input type="text" value="true"/>	<input type="text"/>	<input type="button" value="v"/>

- e After saving, click the **Cancel** button.
- f In the Object Name field type **Change** and press **Enter**.

- g** Select the Data Policy tab and ensure that:
- ▶ The **header,coordinator** attribute is not empty (its value should be **Coordinator**). Change the value to **false**.

Field Name	API Caption	Exclude	API Data Type
header,company	Company	false	
header,coord.date		true	
header,coord.dept		true	
header,coord.phone	CoordinatorPhone	false	
header,coordinator	Coordinator	false	

- ▶ The **header,orig.operator** attribute is not empty (its value should be **OpenedBy**). Change the value to **false**.
- h** Save the changes.
- i** Restart ServiceCenter: Select **Start > Programs > ServiceCenter 6.2 > Server > Console** to open the ServiceCenter Console.



- j** Click **Stop** and then **Start**.
- k** Continue to "Add an Attribute to the UCMDB Model" on page 234.

3 Configure HP Service Manager 7.0/7.1

If you are connecting to HP Service Manager 7.0/7.1, perform the following procedure. If you are connecting to HP ServiceCenter 6.2, skip this step.

- a Import the unload file relevant to the Service Manager version with which you are working: **ucmdbIntegration7_0x.unl** or **ucmdbIntegration7_1x.unl**. To do so, in Service Manager, click **Menu Navigation > Tailoring > Database Manager**.
 - Right-click the detail button and select **Import/Load**.
 - In the HP Service Manager File Load/Import page, click **Specify File** and browse to the following unload file:
C:\hp\UCMDBServer\runtime\fcmdb\CodeBase\ServiceManagerAdapter7-1
The file is loaded via the file browser.
 - Enter the description in the **Import Description** box.
 - Select **winnt** in the **File Type** list.
 - Select a display option.
 - Click **Load FG** to start loading.
- b Continue with "Add an Attribute to the UCMDB Model" on page 234.

Add an Attribute to the ServiceCenter/Service Manager CIT

This section explains how to retrieve additional data from ServiceCenter or Service Manager by adding an attribute to the CIT.

This task includes the following steps:

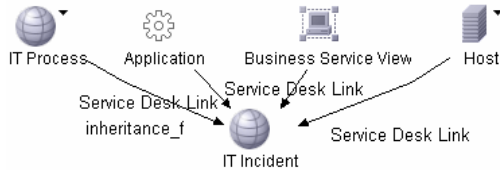
- "Add an Attribute to the UCMDB Model" on page 234
- "Export Attributes from HP ServiceCenter by Changing the Configuration" on page 235

- "Export Attributes from HP Service Manager by Changing the Configuration" on page 237
- "Modify the Adapter Configuration File" on page 240

1 Add an Attribute to the UCMDB Model

Edit the Incident CIT to add the new attribute to UCMDB as follows:

- a Navigate to **Modeling > CI Type Manager**.
- b In the CI Types pane, select **IT Process > IT Incident**.

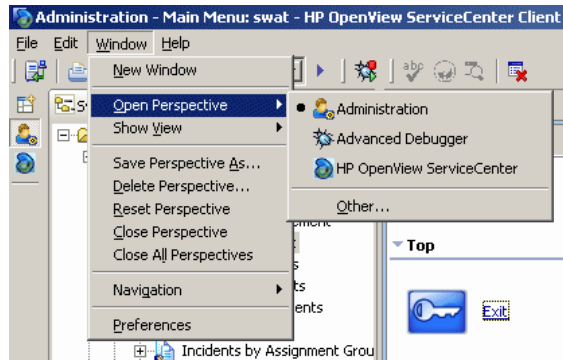


- c Select the **Attributes** tab and add the new attribute.
- d Continue to "Export Attributes from HP ServiceCenter by Changing the Configuration" on page 235 or "Export Attributes from HP Service Manager by Changing the Configuration" on page 237.

2 Export Attributes from HP ServiceCenter by Changing the Configuration

If you are connecting to HP ServiceCenter, perform the following procedure.

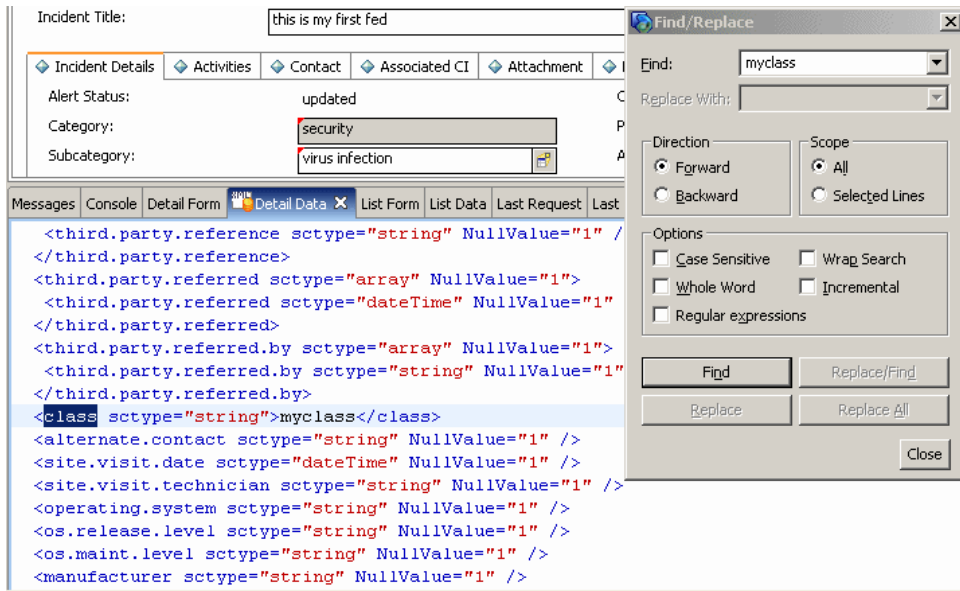
- a In HP ServiceCenter, open the ServiceCenter client.
- b Select **Window > Open Perspective > Administration:**



- c Select **Incident Management > All Open Incidents**, and select one of the incidents you created.

Note: Verify that the value in the Class field is the one that you want to report to UCMDB.

- d Search for the value you entered in the Class field (that is, **myclass**), in the XML file displayed below. This is the CI name in ServiceCenter.



- e Display **WSDL Configuration** in the Navigator (**Main Menu > Menu navigation > Toolkit**). Locate the Object Name field, enter **Incident** and press **Enter**.
- f Select the **Data Policy** tab. Enter a name for the CI mentioned in the XML file (that is, **class**). Change the value to **false**. Save your changes.
- g Restart ServiceCenter: Select **Start > Programs > ServiceCenter 6.2 > Server > Console** to open the ServiceCenter Console.
- h Click **Stop** and then **Start**.
- i Continue to "Modify the Adapter Configuration File" on page 240.

- b Open one of the incidents you created: Select **Incident Management > Search Incidents**. Click the search button (you can filter the fields to limit the search).

The screenshot shows the HP Universal CMDB interface for updating incident IM10002. At the top, there is a table with columns: Incident..., Open Time, Update Time, Alert Status, Category, and Brief Description. Below the table is a toolbar with buttons for OK, Cancel, Previous, Next, Save, Undo, Close, Find, Fill, Clocks, and Apply Template. The main form contains fields for Incident Number (IM10002), Ticket Status (Open), and Incident Title (test1). Below these are tabs for Incident Details, Activities, Contact, CIs and Services, Attachment, History, Alerts, and Related Records. The Incident Details tab is active, showing fields for Alert Status (open), Category (network), Subcategory (remote communications), Product Type (remote communications), Problem Type (dial-in), Manufacturer (Unknown), Class (myclass), Owner (falcon), Primary Asgn Group (LAN SUPPORT), Assignee Name, Second Asgn Group (TELCOM SUPPORT), Hot Ticket (unchecked), Total Loss of Service (unchecked), Initial Impact Assessment (1 - Enterprise), and Urgency (4 - Low). At the bottom, there is a Messages console showing XML data for the incident.

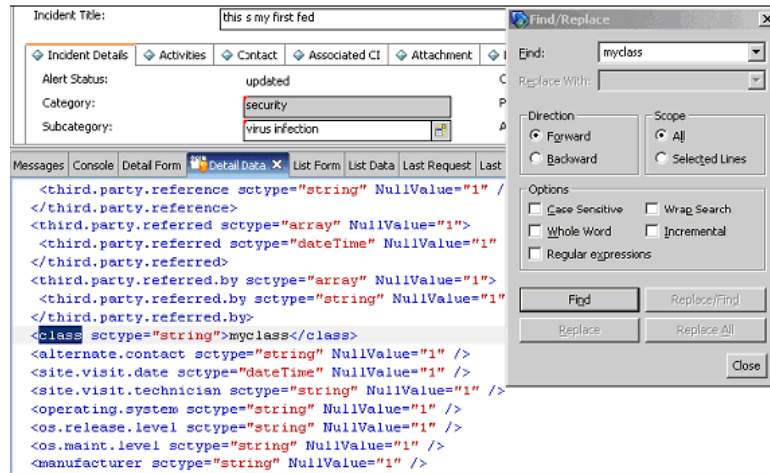
```

<model name="probsummary" query="true">
  <keys>
    <number sctype="string">IM10002</number>
  </keys>
  <instance recordid="IM10002 - test1" uniquequery="number=&quot;IM10002&quot;">
    <number type="string">IM10002</number>
    <number.vj type="string">IM10002</number.vj>
    <number.vj.alerts type="string">IM10002</number.vj.alerts>
    <vj.number.1 type="string">IM10002</vj.number.1>
  </instance>
</model>

```

Note: Verify that the value in the Class field is the one that you want to report to HP Universal CMDB.

- c Search for the value you entered in the Class field (that is, **myclass**), in the XML file displayed below. This is the CI name in Service Manager.



- d Display **WSDL Configuration** in the Navigator (**Main Menu > Menu Navigation > Tailoring**). Locate the Object Name field, enter **UcmdbIncident** and press **Enter**.
- e Select the **Data Policy** tab.
- f Select the **Fields** tab and ensure that the CI name mentioned in the XML file (that is, **class**) appears in the Field list with **ClassName** as its caption. If this attribute does not appear in the Field list, add it and save your changes.
- g Continue to "Modify the Adapter Configuration File" on page 240.

4 Modify the Adapter Configuration File

Perform this procedure for all configurations.

- 1 Navigate to **Data Flow Management > Adapter Management** and select the **ServiceManagerAdapter** that corresponds to your version of Service Manager. Continue and select **Configuration Files > ServiceDeskConfiguration.xml**.
 - a Edit the **ServiceDeskConfiguration.xml** file by navigating to **Data Flow Management > Adapter Management > ServiceManagerAdapter** (the one that corresponds to your version of Service Manager) > **Configuration Files > ServiceDeskConfiguration.xml**
 - b Add the new attribute line under the Incident area: Locate the following marker:

```
<ucmdbClassConfiguration ucmdbClassName="it_incident">  
<attributeMappings>
```

- c Add the following line:

```
<attributeMapping ucmdbAttributeName="incident_class"  
ServiceDeskAttributeName="ClassName"/>
```

where:

- **ucmdbAttributeName="incident_class"** is the value defined in the CI Type Manager
 - **ServiceDeskAttributeName="ClassName"** is the valued defined in ServiceCenter/Service Manager
- d Click **Save**.

Communicate with Service Manager over SSL

The following procedure explains how to open communication with Service Manager over SSL.

This task includes the following steps:

- "Add an SM Self-signed Certificate to the UCMDB Trusted Stores" on page 241
- "Add the SM External Data Source Using Communication Over SSL" on page 242

1 Add an SM Self-signed Certificate to the UCMDB Trusted Stores

- a** Copy the SM self-signed certificate to a directory. (To export SM self-signed certificates, refer to the Service Manager documentation).
- b** Locate the JRE security folder, by default located in:
C:\hp\UCMDB\UCMDBServer\bin\jre\lib
- c** Backup the **cacerts** file by renaming it.
- d** Open a command line window and execute the following commands (to import the previously created or copied certificate):

```
cd C:\hp\UCMDB\UCMDBServer\jre\bin"  
keytool.exe -import -keystore  
C:\hp\UCMDB\UCMDBServer\j2f\JRE\lib\security\cacerts" -trustcacerts -file  
<full path to SM self-signed certificate>
```

- e** Restart the UCMDB service.

2 Add the SM External Data Source Using Communication Over SSL

- a Locate the `adapter\service_desk_adapter.xml` file in the `C:\hp\UCMDB\UCMDBServer\root\lib\packages\serviceDeskAdapter.zip`.
- b Add an `url` field to the `parameters` element:

```
<parameters>
  <parameter name="credentialsId" description="Special type of property,
handled by UCMDB for credentials menu" type="string" mandatory="true" />
  <parameter name="host" description="The remote machine's hostname or ip
this datastore will connect to" 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">13080</parameter>
  <parameter name="url" description="The SSL URL" type="integer" display-
name="SSL URL" mandatory="false" order-index="12"></parameter>
</parameters>
```

- c Click **Save**.
- d In UCMDB, navigate to **Data Flow Management > Integration Studio**.
- e Refresh the configuration by clicking the **Refresh** button. For details, see "Integration Points Pane" in *HP Universal CMDB Data Flow Management Guide*.
- f Define an integration point using the following parameters: In the Create New Integration Point dialog box, choose the **ServiceDeskAdapter** for your version of ServiceCenter or Service Manager, and enter the user name, password, and URL. The URL field should contain: **https://<SM server name>:13443/sc62server/ws**.

For details, see "Create New Integration Point/Edit Integration Properties Dialog Box" in the *HP Universal CMDB Data Flow Management Guide*.



9

Mapping Between CIT Attributes and Database Tables

This chapter includes:

Tasks

- ▶ Use Eclipse to Map Between CIT Attributes and Database Tables on page 243

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

Tasks

Use Eclipse to Map Between CIT Attributes and Database Tables

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

- ▶ Enable graphical mapping between CMDB 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 UCMDDB server and to view the errors, as a further correctness check.
- Define a sample query on the CMDB server and run it directly from Eclipse, to test the mapping file.

This task includes the following steps:

- "Prerequisites" on page 244
- "Installation" on page 245
- "Prepare the Work Environment" on page 245
- "Create an Adapter" on page 249
- "Configure the CMDB Plugin" on page 249
- "Import the CMDB Class Model" on page 251
- "Build the ORM File – Map CMDB Classes to Database Tables" on page 252
- "Map IDs" on page 254
- "Map Attributes" on page 255
- "Map a Valid Link" on page 256
- "Build the ORM File – Use Secondary Tables" on page 258
- "Define a Secondary Table" on page 259
- "Map an Attribute to a Secondary Table" on page 259
- "Use an Existing ORM File as a Base" on page 259
- "Check the Correctness of the ORM File – Built-in Correctness Check" on page 261
- "Create a New Data Store" on page 261
- "Deploy the ORM File to the CMDB" on page 261
- "Run a Sample TQL" on page 262

1 Prerequisites

Install **Java Runtime Environment (JRE) 6 Update 7** on the machine where you will run Eclipse from the following site:

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

The procedure works with Java 5 (or later) runtime environment.

2 Installation

- a Download and extract **Eclipse IDE for Java EE Developers** from <http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/ganymede/SR1/eclipse-jee-ganymede-SR1-win32.zip> to a local folder, for example, **C:\Program Files\eclipse**.
- b Copy **com.hp.plugin.import_cmdb_model_1.0.jar** to **C:\Program Files\Eclipse\plugins**.

You access this file from the following Sharepoint site:

<http://teams1.sharepoint.hp.com/teams/uCMDDBF-adapters/default.aspx?RootFolder=%2fteams%2fuCMDDBF%2dadapters%2fShared%20Documents%2fGDBA%20UI%20%28eclipse%20based%29%2e%20Compatible%20starting%20uCMDDB%207%2e5%2e1&FolderCTID=&View=%7b8BE99EE7%2d790B%2d47BB%2d881A%2dFF51F942BAA9%7d>

- c Launch **C:\Program Files\Eclipse\eclipse.exe** (requires at least a Java 5 runtime environment). 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.

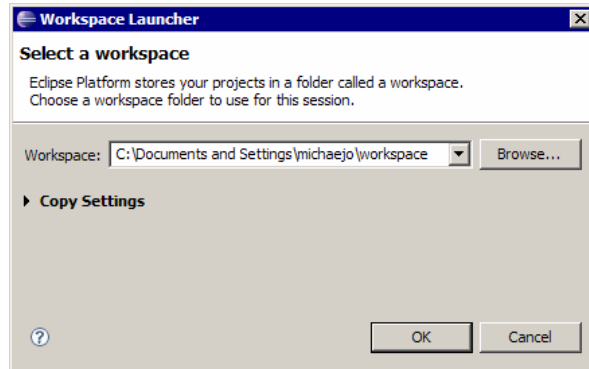
- a Extract the file **workspaces_gdb.rar** into **C:\Documents and Settings\All Users\workspaces**.

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.

You access this file from the following Sharepoint site:

<http://teams1.sharepoint.hp.com/teams/uCMDBF-adapters/default.aspx?RootFolder=%2fteams%2fuCMDBF%2dadapters%2fShared%20Documents%2fGDBA%20UI%20%28eclipse%20based%29%2e%20Compatible%20starting%20uCMDB%207%2e5%2e1&FolderCTID=&View=%7b8BE99EE7%2d790B%2d47BB%2d881A%2dFF51F942BAA9%7d>

- b** In Eclipse, choose **File > Switch Workspace > Other:**

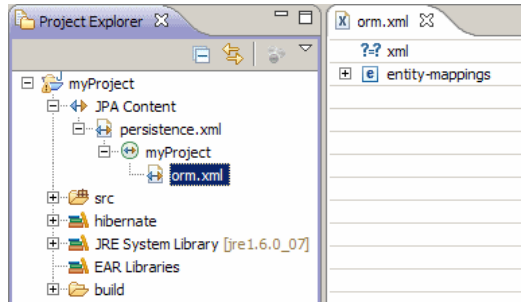


If you are working with:

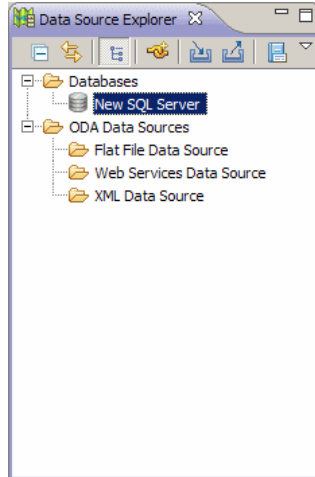
- ▶ SQL Server, select the following folder: **C:\Documents and Settings\All Users\workspace_gdb_sqlserver.**
- ▶ MySQL, select the following folder: **C:\Documents and Settings\All Users\workspace_gdb_mysql.**
- ▶ Oracle, select the following folder: **C:\Documents and Settings\All Users\workspace_gdb_oracle.**

- c** Click **OK**.

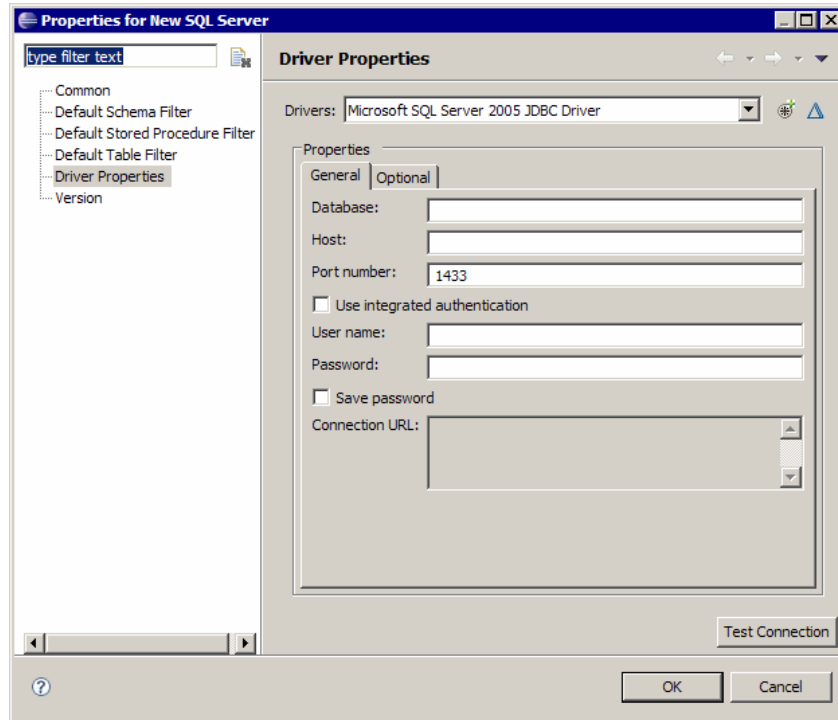
- d** In Eclipse, display the Project Explorer view and select **<Active project>** **> JPA Content > persistence.xml > <active project name > > orm.xml**.



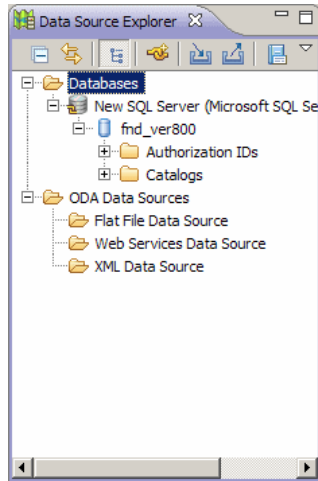
- e** In the Data Source Explorer view (the bottom left pane), right-click the database connection and select the **Properties** menu.



- 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.

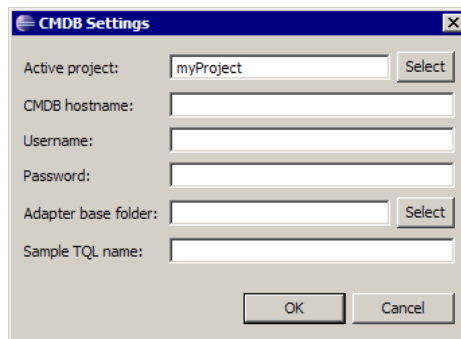


4 Create an Adapter

Create an adapter using the guidelines in "Create a Database Adapter – Minimal Method" or "Create a Database Adapter – Advanced Method" in the *HP Universal CMDB Data Flow Management Guide*.

5 Configure the CMDB Plugin

- a In Eclipse, click **UCMDB > Settings** to open the **CMDB Settings** dialog box:

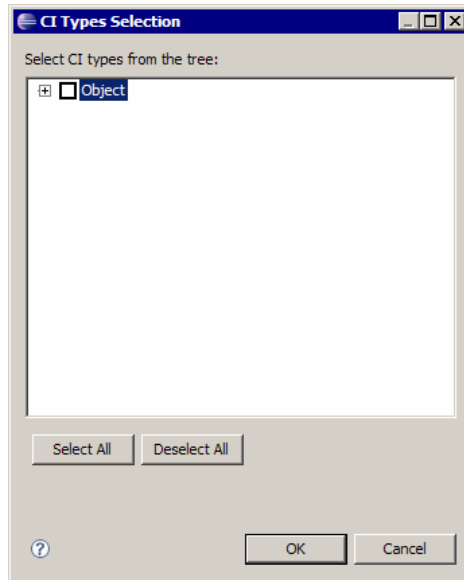


- b** If not already selected, select the newly created JPA project as the Active project.
- c** Enter the CMDB host name, for example, **localhost** or **labm1.itdep1**. There is no need to include the port number or **http://** prefix in the address.
- d** Fill in the user name and password for accessing the CMDB API, usually **admin/admin**.
- e** Make sure that the **C:\hp** folder on the UCMDB 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 **C:\UCMDB\UCMDBServer\j2f\fcmdb\CodeBase\<adapter name>**. Verify that there is no backslash (\) at the end.

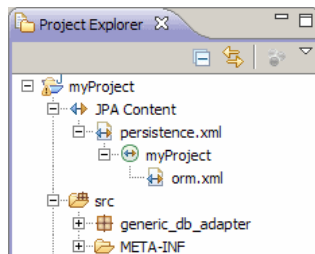
6 Import the CMDB Class Model

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

- a Click **UCMDB > Import CMDB Class Model** to open the **CI Type Selection** dialog box:



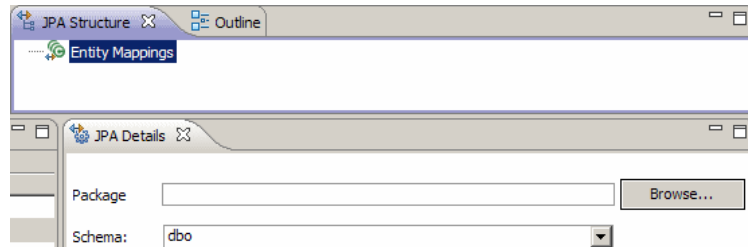
- b Select the CI types that you intend to map as JPA entities. Click **OK**. The CI types are imported as Java classes. Verify that they appear under the **src** folder of the active project:



7 Build the ORM File – Map CMDDB Classes to Database Tables

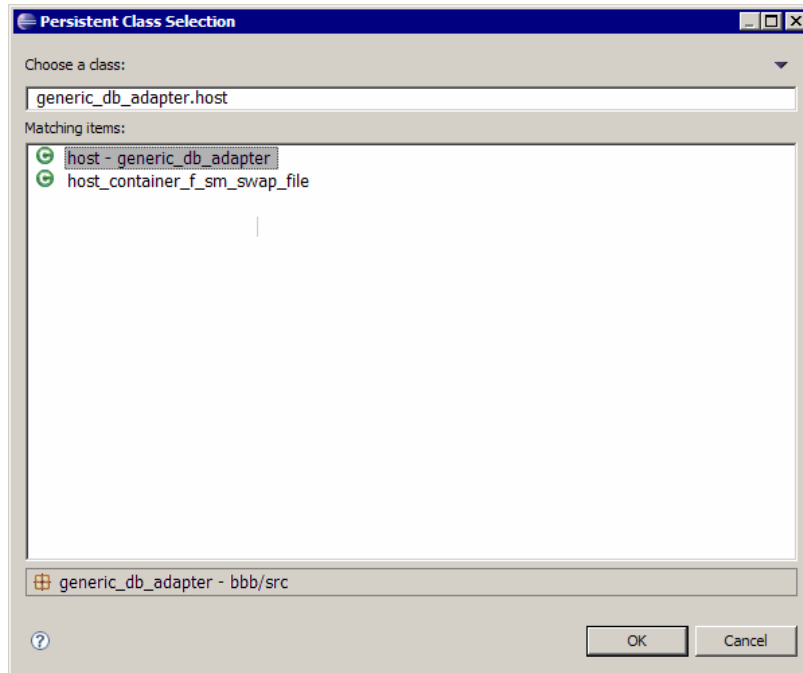
In this step, you map the Java classes (that you imported in the previous step) to the database tables.

- 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**).

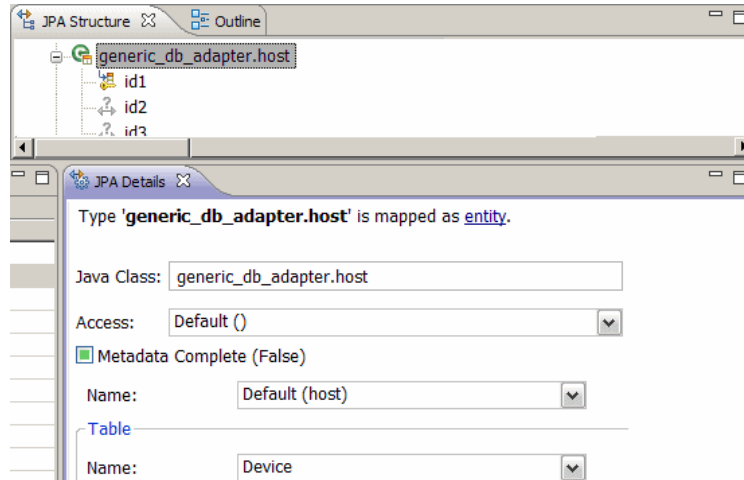
- c Click **Browse** and select the CMDB class to be mapped (all CMDB classes belong to the **generic_db_adapter** package).



- 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**.

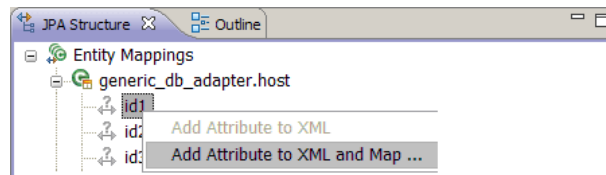
- e In the JPA Details view, select the primary database table to which the CMDB class should be mapped. Leave all other fields unchanged.



8 Map IDs

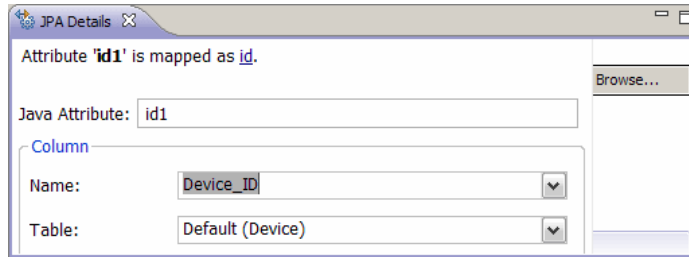
According to JPA standards, each persistent class must have at least one ID attribute. For CMDB classes, you can map up to three attributes as IDs. Potential ID attributes are called **id1**, **id2**, and **id3**. To map an ID attribute:

- 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

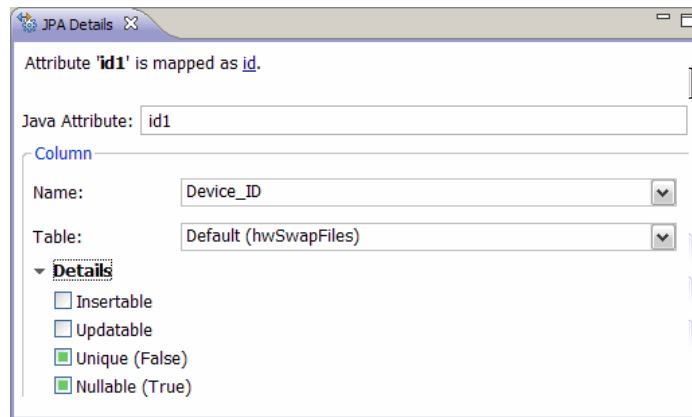
In this step, you map attributes to the database columns.

- 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

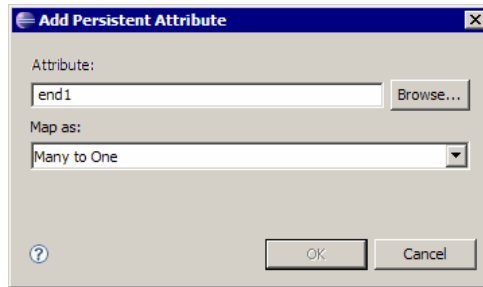
Perform the steps described in step b on page 252 for mapping a CMDB class denoting a valid link. The name of each such class takes the following structure: **<end1 entity name>_<link name>_<end 2 entity name>**. For example, a **Contains** link between a host and a location is denoted by a Java class whose name is **generic_db_adapter.host_contains_location**. For details, see "The reconciliation_rules.txt File" in the *HP Universal CMDB Data Flow Management Guide*.

- a Map the ID attributes of the link class as described in "Map IDs" on page 254. 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 **Updatable** check boxes and click **OK**.

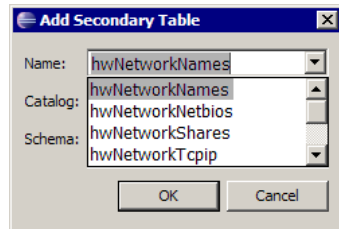
- If the **end1/end2** target entity has more than one ID, click the **Add** button to add additional join columns and map them in the same way as described in the previous step.

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:

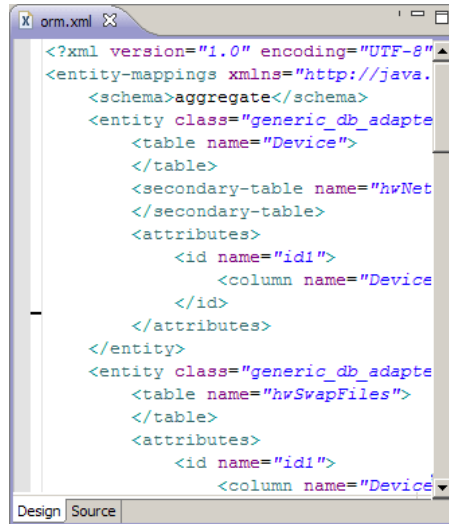
- a Map the attribute as described in "Map Attributes" on page 255.
- b In the **Column** section of the JPA Details view, select the secondary table name in the **Table** field, to replace the default value.

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.



```

<?xml version="1.0" encoding="UTF-8"
<entity-mappings xmlns="http://java.
<schema>aggregate</schema>
<entity class="generic_db_adapte
<table name="Device">
</table>
<secondary-table name="hwNet
</secondary-table>
<attributes>
<id name="idi">
<column name="Device
</id>
</attributes>
</entity>
<entity class="generic_db_adapte
<table name="hwSwapFiles">
</table>
<attributes>
<id name="idi">
<column name="Device

```

- d Paste all copied entity mapping 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 in step b on page 252. 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

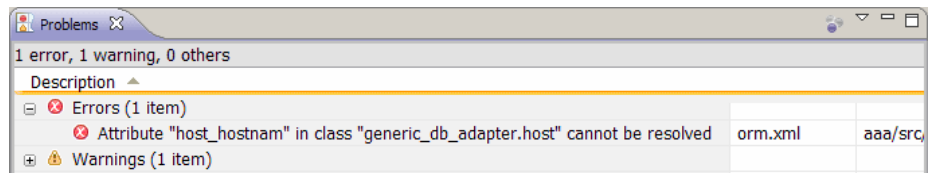
If an adapter already exists, the Eclipse Plugin can be used to edit its ORM file graphically. Import the ORM file into Eclipse, edit it using the plugin and then deploy it back to the UCMDDB machine. To import the ORM file, press the button on the Eclipse toolbar. A confirmation dialog is displayed. Click **OK**. The ORM file is copied from the UCMDDB machine to the active Eclipse project and all relevant classes are imported from the UCMDDB 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 UCMDB machine as described in "Deploy the ORM File to the CMDB" on page 261.

16 Check the Correctness of the ORM 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 Data Store

If no data store exists in the CMDB for this adapter, you can create it by clicking **UCMDB > Create Data Store**. Fill in the data store name in the dialog box that opens. The orm.xml file is copied to the adapter folder. A data store 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" on page 225.

18 Deploy the ORM File to the CMDB

Save the orm.xml file and deploy it to the CMDB server: by clicking **UCMDB > Deploy ORM**. The orm.xml 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 data store has yet been defined using the adapter, no mapping errors are detected upon deployment.

19 Run a Sample TQL

- a** Define a query using Query Manager and not View Manager.
- b** Create a data store using the **GenericDBAdapter** adapter. For details, see "Generic Database Adapter" on page 151 and "Create New Integration Point/Edit Integration Properties Dialog Box" in the *HP Universal CMDB 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 data store.
- d** When configuring the CMDB plugin, use this sample query name in the Settings dialog box. For details, see "Configure the CMDB Plugin" on page 249.
- 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.

Part II

APIs

10

Introduction to APIs

This chapter includes:

Concepts

- APIs Overview on page 265

Concepts

APIs Overview

The following APIs are included with HP Universal CMDB:

- **UCMDB Web Service API.** Enables writing configuration item definitions and topological relations to the UCMDB (Universal Configuration Management database), and querying the information with TQL and ad hoc queries. For details, see "HP Universal CMDB Web Service API" on page 267.
- **DFM 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 UCMDB (Universal Configuration Management database). For details, see "HP Universal CMDB API" on page 343.
- **DFM Web Service.** Explains how third-party or custom tools can use the HP Discovery and Dependency Mapping Web Service to manage DFM. For details, see "Data Flow Management Web Service API" on page 355.

11

HP Universal CMDB Web Service API

This chapter includes:

Concepts

- ▶ Conventions on page 268
- ▶ Using the HP Universal CMDB Web Service API on page 268
- ▶ HP Universal CMDB Web Service API Reference on page 270
- ▶ Returning Unambiguous Topology Map Elements on page 270

Tasks

- ▶ Call the Web Service on page 274
- ▶ Query the UCMDB on page 274
- ▶ Update the UCMDB on page 279
- ▶ Query the UCMDB Class Model on page 281
- ▶ Query for Impact Analysis on page 283

Reference

- ▶ UCMDB Query Methods on page 283
- ▶ UCMDB Update Methods on page 298
- ▶ UCMDB Impact Analysis Methods on page 301
- ▶ Use Cases on page 304
- ▶ Examples on page 305
- ▶ UCMDB General Parameters on page 336
- ▶ UCMDB Output Parameters on page 340

Concepts

Conventions

This chapter uses the following conventions:

- ▶ **UCMDB** refers to the Universal Configuration Management database itself. **HP Universal CMDB** refers to the application.
- ▶ UCMDB 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.

Using the HP Universal CMDB Web Service API

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 Universal CMDB (UCMDB). The API provides methods to:

- ▶ add, remove, and update CIs and relations in the CMDB
- ▶ retrieve information about the class model
- ▶ retrieve impact analyses
- ▶ 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 Web Service API should be familiar with:

- ▶ The SOAP specification
- ▶ An object-oriented programming language such as C++, C# or Java
- ▶ HP Universal CMDB

This section includes the following topics:

- "Uses of the API" on page 269
- "Permissions" on page 269

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 UCMDB 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 UCMDB to create a central UCMDB 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 UCMDB to take advantage of the UCMDB query capabilities.
- Other systems, such as the Change Control Management (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 Business Service Management:

- **HP Universal CMDB standalone.** Log in using the credentials of a UCMDB user who has been granted permissions on the discovery and integration resources.

For details, see "Security Manager Page" in the *HP UCMDB Administration Guide*.

- ▶ **HP Universal CMDB embedded in Business Service Management.** Log in using the credentials of a Business Service Management user. The user must have been granted the relevant permissions on the HP Universal CMDB resource in Business Service Management.

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:

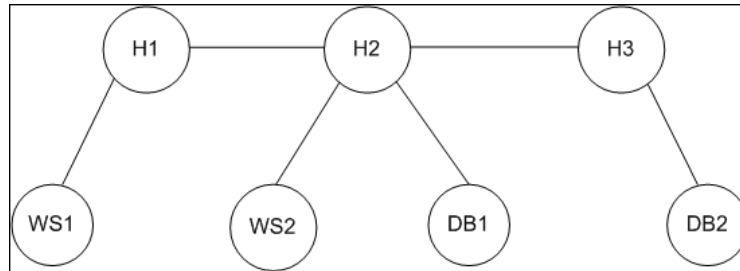
```
\\<HP Universal CMDB root directory>\UCMDBServer\j2f\AppServer\  
webapps\site.war\amdocs\eng\doc_lib\Integrations\CMDB_Schema\  
webframe.html
```

Returning Unambiguous Topology Map Elements

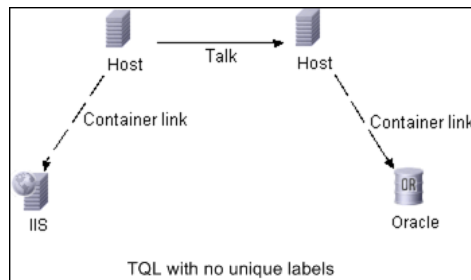
Query methods that return the data in topology or topologyMap elements search the system for a match of a TQL query. The following diagrams illustrate how the resulting topology and topologyMap structures are affected by the use of unique labels in the query.

Labels are user-specified names in the query for relations and configuration items in specific configurations. The labels specified in the query are used as the node labels in the returned map. If no labels are specified, the CI or Relation Type Name is used as the label in the resulting map. The following example illustrates specifying labels IISHost and DBHost in place of the default Host label, and labels ContainerIIS and ContainsDB in place of the default Container Link label.

The following example represents a small IT universe model. There are three hosts: H1, H2, H3, which host Web servers (WS) and database managers (DB). WS1 resides on H1. DB1 and WS2 both reside on H2. DB2 resides on H3.



This query is defined using the default labels:



The result of running this TQL query on the IT universe can be a **Topology** or **TopologyMap** element.

Topology Response

CIs: H1, H2, H3, WS1, WS2, DB1, DB2

Relations: H1-WS1, H1-H2, H2-H3, WS2-H2, DB1-H2, DB2-H3

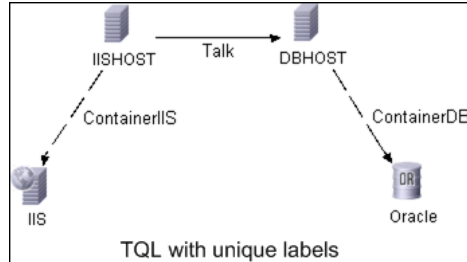
TopologyMap Response

```
CINode:  
  label: Host  
  CIs: H1, H2  
  
CINode:  
  label: Host  
  CIs: H2, H3  
  
CINode:  
  label: DB  
  CIs: DB1, DB2  
  
CINode:  
  label: Webserver  
  CIs: IIS  
  
relationNode:  
  label: talk  
  relations: H1-H2, H2-H3  
  
relationNode:  
  label: Container Link  
  relations: WS1-H1, WS2-H2  
  
relationNode:  
  label: Container Link  
  relations: DB2-H3, DB1-H2
```

In the above TopologyMap response, the first two CINodes contain identical Host labels, corresponding to the two Host CIs in the query. Both of these CINodes contain host H2, with no indication of why H2 is duplicated.

The last two relationNodes contain identical Contained labels, corresponding to the two Container link relations in the query.

The duplications occur because no unique labels are specified in the query, resulting in the use of default labels (the type names `Host` and `Container`) in the map. To extract a more usable map, define queries with unique labels for each configuration to be matched, as shown in the following query:



The topology result is identical to that of the TQL without unique labels. The `topologyMap` result, however, is different: Each label is now unique.

```

CINode:
  label: IISHOST
  CIs: H1, H2

CINode:
  label: DBHOST
  CIs: H2, H3

...

relationNode:
  label: ContainerIIS
  relations: WS1-H1, WS2-H2

relationNode:
  label: ContainerDB
  relations: DB2-H3, DB1-H2

```

In this map, it is clear why H2 is returned twice. The unique labels indicate that it is returned once as a Web server host and once as a database host.

Tip: Wherever possible in the UCMDB, apply unique, user-defined labels to specific configurations.

Tasks

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, the UCMDDB 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](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](http://<server>[:port]/axis2/services/UcmdbService)

For examples of connecting to the UCMDDB, see "Use Cases" on page 304.

Query the UCMDDB

The UCMDDB is queried using the APIs described in "UCMDDB Query Methods" on page 283.

The queries and the returned UCMDDB elements always contain real UCMDDB IDs.

For examples of the use of the query methods, see "Query Example" on page 309.

This section includes the following topics:

- ▶ "Just In Time Response Calculation" on page 275
- ▶ "Processing Large Responses" on page 275

- "Specifying Properties to Return" on page 276
- "Concrete Properties" on page 277
- "Derived Properties" on page 277
- "Naming Properties" on page 278
- "Other Property Specification Elements" on page 278

Just In Time Response Calculation

For all query methods, the UCMDB 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 296 and "releaseChunks" on page 297.

Specifying Properties to Return

CI 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 UCMDB 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 UCMDB 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 UCMDB 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 UCMDB 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 UCMDB

You update the UCMDB with the update APIs. For details of the API methods, see "UCMDB Update Methods" on page 298.

For examples of the use of the update methods, see "Update Example" on page 326.

This task includes the following steps:

- "UCMDB Update Parameters" on page 279
- "Use of ID Types with Update Methods" on page 280
- "UCMDB Update Methods" on page 298

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 UCMDB. 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 UCMDB 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 the "CIsAndRelationsUpdates" on page 279.

Inserting New Configuration Items into the UCMDB

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 UCMDB ID in the structure `createIDsMap` because there is no `clientID`. For details, see "addCIsAndRelations" on page 298 and "UCMDB Query Methods" on page 283.

Inserting New Relations into the UCMDB

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 UCMDB, then those CIs that already exist must be identified by a real UCMDB ID or be specified in a `referencedCIs` collection.

Query the UCMDB 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 330.

This section provides information on the following methods that return information about CITs and relations:

- "getClassAncestors" on page 281
- "getAllClassesHierarchy" on page 282
- "getCmdbClassDefinition" on page 282

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 336.
className	The type name. For details, see "Type Name" on page 338.

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
<code>cmdbContext</code>	For details, see "CmdbContext" on page 336.

Output

Parameter	Comment
<code>classesHierarchy</code>	A collection of pairs of class name and parent class name.
<code>comments</code>	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
<code>cmdbContext</code>	For details, see "CmdbContext" on page 336.
<code>className</code>	The type name. For details, see "Type Name" on page 338.

Output

Parameter	Comment
cmdbClass	The class definition, consisting of name, classType, displayLabel, description, parentName, qualifiers, and attributes.
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 332.

Reference

UCMDB Query Methods

This section provides information on the following methods:

- "executeTopologyQueryByName" on page 284
- "executeTopologyQueryByNameWithParameters" on page 284
- "executeTopologyQueryWithParameters" on page 285
- "getChangedCIs" on page 286
- "getCINeighbours" on page 287
- "getCIsByID" on page 288
- "getCIsByType" on page 289
- "getFilteredCIsByType" on page 290
- "getQueryNameOfView" on page 294

- "getTopologyQueryExistingResultByName" on page 295
- "getTopologyQueryResultCountByName" on page 295
- "pullTopologyMapChunks" on page 296
- "releaseChunks" on page 297

executeTopologyQueryByName

The `executeTopologyQueryByName` method retrieves the topology map that matches the specified query.

Tip: The map contains more information and is easier to understand if the label for each `CINode` and each `relationNode` in the TQL is unique. For details, see "Returning Unambiguous Topology Map Elements" on page 270.

Input

Parameter	Comment
<code>cmdbContext</code>	For details, see "CmdbContext" on page 336.
<code>queryName</code>	The name of the TQL in the UCMDB with which to retrieve the map.
<code>queryTypedProperties</code>	A collection of sets of properties to retrieve to items of a specific Configuration Item Type.

Output

Parameter	Comment
<code>topologyMap</code>	For details, see "TopologyMap" on page 341.

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 336.
<code>queryName</code>	The name of the parameterized TQL in the UCMDB 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
<code>topologyMap</code>	For details, see "TopologyMap" on page 341.
<code>chunkInfo</code>	For details, see: "ChunkInfo" on page 341, "Processing Large Responses" on page 275.

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 UCMDB. You can use this method when you do not have access to the UCMDB user interface to define a query, or when you do not want to save the query to the database.

Input

Parameter	Comment
cmdbContext	For details, see "CmdbContext" on page 336.
queryXML	An XML representation of a TQL.
parameterizedNodes	The conditions each node must meet to be included in the query results.

Output

Parameter	Comment
topologyMap	For details, see "TopologyMap" on page 341.
chunkInfo	For details, see "ChunkInfo" on page 341 and "Processing Large Responses" on page 275.

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 336.
ids	The list of the IDs of the root CIs whose related CIs are checked for changes. Only real UCMDB 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 ChangedDataInfo elements.

getCI Neighbours

The `getCI Neighbours` 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 336.
ID	The ID of the CI with which to retrieve the neighbors. This must be a real UCMDB ID.
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 338.

Parameter	Comment
CIProperties	The data to be returned on each configuration item, called the Query Layout in the user interface. For details, see "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." on page 278.
relationProperties	The data to be returned on each relation (called the Query Layout in the user interface). For details, see "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." on page 278

Output

Parameter	Comment
topology	For details, see "Topology" on page 340.
comments	For internal use only.

getCIsByID

The `getCIsByID` method retrieves configuration items by their UCMDB IDs.

Input

Parameter	Comment
cmdbContext	For details, see "CmdbContext" on page 336.
CIsTypedProperties	A typed properties collection. For details, see "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." on page 278.
IDs	Only real UCMDB IDs are valid in this collection.

Output

Parameter	Comment
CIs	Collection of CI elements.
chunkInfo	For details, see: "ChunkInfo" on page 341, "Processing Large Responses" on page 275.

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 336.
type	The class name. For details, see "Type Name" on page 338.
properties	The data to be returned on each configuration item. For details, see "CustomProperties" on page 278.

Output

Parameter	Comment
CIs	Collection of CI elements.
chunkInfo	For details, see: "ChunkInfo" on page 341, "Processing Large Responses" on page 275.

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 ChangedDuring 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>

Operator	Type of Condition/Comments
IsNull	String, numerical, and list The item's property has no value. When operator IsNull is used, the value of the condition is ignored, and in some cases can be nil.
Less	Numerical
LessEqual	Numerical
Like	String 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, %Bi% matches Bismark and Bay of Biscay, but not biscuit.
LikeIgnoreCase	String Use the LikeIgnoreCase operator as you use the Like operator. The match, however is not case-sensitive. Therefore, %Bi% matches biscuit.
NotEqual	String and numerical
UnchangedDuring	Date 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. 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. Note: The name UnchangedDuring is kept to preserve backward compatibility. In previous versions, the operator was used only with create and modify time properties.

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`. `samplell` extends the CI with two attributes, `level` and `grade`. This example sets up a query for the properties of `samplell` that were inherited from `sample` by specifying them by name.

```
GetFilteredCIsByType request = new GetFilteredCIsByType()
request.setCmdbContext(cmdbContext)
request.setType("samplell")
CustomProperties customProperties = new CustomProperties();
PropertiesList propertiesList = new PropertiesList();
propertiesList.addPropertyName("name");
propertiesList.addPropertyName("size");
customProperties.setPropertiesList(propertiesList);
request.setProperties(customProperties)
```

Input

Parameter	Comment
cmdbContext	For details, see "CmdbContext" on page 336.
type	The class name. For details, see "Type Name" on page 338. The type can be any of the types defined using the CI Type Manager. For details, see "CI Type Manager" in <i>Modeling Guide</i> .

Parameter	Comment
properties	The data to be returned on each CI. (Called the Query Layout in the user interface) For details, see "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." on page 278.
conditions	A collection of name-value pairs and the operators that relate one to the other. For example, host_hostname like QA.
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 341 and "Processing Large Responses" on page 275.

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 336.
viewName	The name of a view, that is, a sub-set of the class model in the UCMDB.

Output

Parameter	Comment
queryName	The name of the TQL in the UCMDB 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 336.
queryName	The name of a TQL.
queryTypedProperties	A collection of sets of properties to retrieve to items of a specific Configuration Item Type.

Output

Parameter	Comment
queryName	The name of the TQL in the UCMDB 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 336.
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 UCMDB 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 341 and "Query the UCMDB" on page 274.

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 309.

Input

Parameter	Comment
cmdbContext	For details, see "CmdbContext" on page 336.
ChunkRequest	The number of the chunk to retrieve and the ChunkInfo that is returned by the query method.

Output

Parameter	Comment
topologyMap	For details, see "TopologyMap" on page 341.
comments	For internal use only.

Example of Handing Chunks:

```

GetCIsByType request =
    new GetCIsByType(cmdbContext, typeName, customProperties);
GetCIsByTypeResponse response =
    ucmdbService.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.sizeCICollection() ; 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 336.
chunksKey	The identifier of the data on the server that was chunked. The key is an element of ChunkInfo.

UCMDB Update Methods

This section provides information on the following methods:

- "addCIsAndRelations" on page 298
- "addCustomer" on page 299
- "deleteCIsAndRelations" on page 300
- "removeCustomer" on page 300
- "updateCIsAndRelations" on page 300

addCIsAndRelations

The `addCIsAndRelations` method adds or updates CIs and relations.

If the CIs or relations do not exist in UCMDB, they are added and their properties are set according to the contents of the `CIsAndRelationsUpdates` argument.

If the CIs or relations do exist in UCMDB, 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 UCMDB IDs.

Input

Parameter	Comment
<code>cmdbContext</code>	For details, see "CmdbContext" on page 336.
<code>updateExisting</code>	Set to <i>true</i> to update items that already exist in the UCMDB. 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 279.
<code>ignoreValidation</code>	If is true, no check is performed before updating the uCMDB.

Output

Parameter	Comment
<code>CreatedIDsMap</code>	The map of client IDs to UCMDB IDs. For details, see "addCIsAndRelations" on page 298.
<code>comments</code>	For internal use only.

addCustomer

The `addCustomer` method adds a customer.

Input

Parameter	Comment
<code>CustomerID</code>	The numeric ID of the customer.

deleteCIsAndRelations

The `deleteCIsAndRelations` method removes the specified configuration items and relations from the UCMDB.

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
<code>cmdbContext</code>	For details, see "CmdbContext" on page 336.
<code>CIsAndRelationsUpdates</code>	The items to delete. For details, see "CIsAndRelationsUpdates" on page 279

removeCustomer

The `removeCustomer` method deletes a customer record.

Input

Parameter	Comment
<code>CustomerID</code>	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 UCMDB, an exception is thrown.

`CreatedIDsMap` is a map or dictionary of type `ClientIDToCmdbID` that connects the client's temporary IDs with the corresponding real UCMDB IDs.

Input

Parameter	Comment
cmdbContext	For details, see "CmdbContext" on page 336.
CIsAndRelationsUpdates	The items to update. For details, see "CIsAndRelationsUpdates" on page 279.
ignoreValidation	If true, no check is performed before updating the uCMDB.

Output

Parameter	Comment
CreatedIDsMap	The map of client IDs to UCMDB IDs. For details, see "addCIsAndRelations" on page 298.

 **UCMDB Impact Analysis Methods**

This section provides information on the following methods:

- "calculateImpact" on page 301
- "getImpactPath" on page 302
- "getImpactRulesByNamePrefix" on page 303

 **calculateImpact**

The calculateImpact method calculates which CIs are affected by a given CI according to the rules defined in the UCMDB.

This shows the effect of an event triggering of the rule. The identifier output of calculateImpact is used as input for getImpactPath.

Input

Parameter	Comment
cmdbContext	For details, see "CmdbContext" on page 336.
impactCategory	The type of event that would trigger the rule being simulated.
IDs	A collection of ID elements.
impactRulesNames	A collection of ImpactRuleName elements.
severity	The severity of the triggering event.

Output

Parameter	Comment
impactTopology	For details, see "Topology" on page 340.
identifier	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` is used as the `identifier` input argument of `getImpactPath`.

Input

Parameter	Comment
cmdbContext	For details, see "CmdbContext" on page 336.
identifier	The key to the server response that was returned by <code>calculateImpact</code> .
relation	A Relation based on one of the ShallowRelations returned by <code>calculateImpact</code> in the <code>impactTopology</code> element.

Output

Parameter	Comment
impactPathTopology	A CIs collection and an ImpactRelations 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 336.
ruleNamePrefixFilter	A string containing the first letters of the rule names to match.

Output

Parameter	Comment
impactRules	impactRules is composed of zero or more impactRule. An impactRule, which specifies the effect of a change, is composed of ruleName, description, queryName, and isActive.

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 UCMDB" on page 304
- "Querying the UCMDB" on page 304
- "Querying the Class Model" on page 305
- "Analyzing Change Impact" on page 305

Populating the UCMDB

Use cases:

- A third-party asset management updates the UCMDB with information available only in asset management
- A number of third-party systems populate the UCMDB 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 CMDB query capabilities

Querying the UCMDB

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 UCMDB. 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 UCMDB 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" on page 306
- "Query Example" on page 309
- "Update Example" on page 326
- "Class Model Example" on page 330
- "Impact Analysis Example" on page 332

 **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 = 8080;
private static final String FILE = "/axis2/services/UcmdbService";
```

```
protected UcmdbService createUcmdbService
(String username, String password) throws Exception{
    URL url;
    UcmdbServiceStub serviceStub;
```

```
try {
    url = new URL
        (Demo.PROTOCOL, Demo.HOST_NAME,
        Demo.PORT, Demo.FILE);
    serviceStub = new UcmdbServiceStub(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.UcmdbFaultException;
import com.hp.ucmdb.generated.services.UcmdbService;
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 getClsByTypeDemo() {
        GetClsByType request = new GetClsByType();
        //set cmdbcontext
        CmdbContext cmdbContext = getContext();
        request.setCmdbContext(cmdbContext);
        //set Cls type
        request.setType("anyType");
        //set Cls propeties 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.setClsTypedProperties(properties);  
try {  
    GetClsByIdResponse response =  
        getStub().getClsById(request);  
    Cls cis = response.getCls();  
} catch (RemoteException e) {  
    //handle exception  
} catch (UcmdbFaultException e) {  
    //handle exception  
}  
  
}
```

```
public void getFilteredClsByTypeDemo() {  
    GetFilteredClsByType request = new GetFilteredClsByType();  
    CmdbContext cmdbContext = getContext();  
    //set cmdbcontext  
    request.setCmdbContext(cmdbContext);  
    //set Cls 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 (UcmdbFaultException 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 (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 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 getCI NeighboursDemo() {
    GetCI Neighbours request = new GetCI Neighbours();
    //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 properties 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 (UcmdbFaultException 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 (UcmdbFaultException 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())){

```

```

        Cls cisTOAdd = ciNode.getCIs();
        Cls 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 relations2;
}

```

```
private Cls mergeClsGroups(Cls cis1, Cls cis2) {  
    for(int i=0 ; i < cis2.sizeCIList() ; i++) {  
        cis1.addCI(cis2.getCI(i));  
    }  
    return cis1;  
}  
  
}
```

 **Update Example**

```
package com.hp.ucmdb.demo;

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.UcmdbFaultException;
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;

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();
        CI ci = new CI();
        ID id = new ID();
        id.setBase("temp1");
        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.addStrProp(strProp);
props.setStrProps(strProps);
ci.setProps(props);
cis.addCI(ci);
updates.setCIsForUpdate(cis);
request.setCIsAndRelationsUpdates(updates);

```

```

try {
    AddCIsAndRelationsResponse response =
        getStub().addCIsAndRelations(request);
    for(int i = 0 ; i < response.sizeCreatedIDsMapList() ; i++) {
        ClientIDToCmdbID idsMap = response.getCreatedIDsMap(i);
        //do something
    }
} catch (RemoteException e) {
    //handle exception
} catch (UcmdbFaultException e) {
    //handle exception
}
}

```

```

public void getUpdateCIsAndRelationsDemo() {
    UpdateCIsAndRelations request = new UpdateCIsAndRelations();
    request.setCmdbContext(getContext());

```

```

CIsAndRelationsUpdates updates =
    new CIsAndRelationsUpdates();
CIs cis = new CIs();
CI ci = new CI();
ID id = new ID();

```

```

id.setBase("temp1");
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.addStrProp(hostKeyProp);
```

```
StrProp hostOSProp = new StrProp();
hostOSProp.setName("host_os");
String hostOSValue = "winXP";
hostOSProp.setValue(hostOSValue);
strProps.addStrProp(hostOSProp);
```

```
StrProp hostDNSProp = new StrProp();
hostDNSProp.setName("host_dnsname");
String hostDNSValue = "dnsname";
hostDNSProp.setValue(hostDNSValue);
strProps.addStrProp(hostDNSProp);
```

```
props.setStrProps(strProps);
ci.setProps(props);
cis.addCI(ci);
updates.setCIsForUpdate(cis);
request.setCIsAndRelationsUpdates(updates);
```

```
try {
    getStub().updateCIsAndRelations(request);
} catch (RemoteException e) {
    //handle exception
} catch (UcmdbFaultException e) {
    //handle exception
}
}
```



```

public void getDeleteCIsAndRelationsDemo() {
    DeleteCIsAndRelations request =
        new DeleteCIsAndRelations();
    request.setCmdbContext(getContext());
    CIsAndRelationsUpdates updates =
        new CIsAndRelationsUpdates();
    CIs cis = new CIs();
    CI ci = new CI();
    ID id = new ID();
    id.setBase("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.addStrProp(strProp1);
    props.setStrProps(strProps);
    ci.setProps(props);
    cis.addCI(ci);
    updates.setCIsForUpdate(cis);
    request.setCIsAndRelationsUpdates(updates);

```

```

        try {
            getStub().deleteCIsAndRelations(request);
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }
    }
}

```

 **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
    }
}
}

```

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" on page 336
- "ID" on page 337
- "Key Attributes" on page 337
- "ID Types" on page 337
- "CIProperties" on page 338
- "Type Name" on page 338
- "Configuration Item (CI)" on page 339
- "Relation" on page 339

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 UCMDB 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 "getCmdmClassDefinition" on page 282.

ID Types

An ID element can contain a real ID, a temporary ID, or can be empty.

A real ID is a string assigned by the UCMDB that identifies an entity in the database. A temporary ID can be any string that is unique in the current request. An empty ID means no value is assigned.

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 UCMDB. When a temporary ID is passed by the client, if the UCMDB 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.

The real ID of a CI is calculated by the UCMDB based on a combination of the CI's type and key properties. The real ID of a Relation is based on the relations's type, the IDs of the two CIs that are part of the relationship, and the relation's key properties. Therefore, key attribute values must be set during CI or Relation creation. If the key properties values are not specified when creating a CI, there are two possibilities:

- If the CIT includes a RANDOM_GENERATED_ID qualifier, the server generates a unique ID.

- ▶ If the CIT does not have a RANDOM_GENERATED_ID qualifier, an exception is thrown.

For details, see "CI Type Manager" in the *Modeling Guide*.

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)" on page 339 and "Relation" on page 339.

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 UCMDB 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 *Modeling Guide*.

The *props* element is a CIProperties collection. For details, see "CIProperties" on page 338.

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 UCMDB 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 HP UCMDB class from which the relation is instantiated. The type can be any of the relation types defined in the UCMDB. For further information on classes or types, see "Query the UCMDB Class Model" on page 281.

For details, see "CI Type Manager" in *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 page 338.

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" on page 340
- "ShallowRelation" on page 340
- "Topology" on page 340
- "CINode" on page 340
- "RelationNode" on page 341
- "TopologyMap" on page 341
- "ChunkInfo" on page 341

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 UCMDB class from which the relation is instantiated. The type can be any of the relation types defined in the UCMDB.

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" on page 340).
- ▶ relationNodes. This is one or more RelationNode (see "RelationNode" on page 341).

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 275.

12

HP Universal CMDB API

This chapter includes:

Concepts

- ▶ Conventions on page 343
- ▶ Using the HP Universal CMDB API on page 344
- ▶ General Structure of an Application on page 345

Tasks

- ▶ Put the API Jar File in the Classpath on page 346
- ▶ Create an Integration User on page 346

Reference

- ▶ HP Universal CMDB API Reference on page 347
- ▶ Use Cases on page 347
- ▶ Examples on page 349

Concepts

Conventions

This chapter uses the following conventions:

- ▶ **UCMDB** refers to the Universal Configuration Management database itself. **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.

Using the HP Universal CMDB API

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 Universal CMDB (UCMDB). The API provides methods to:

- ▶ add, remove, and update CIs and relations in the CMDB
- ▶ retrieve information about the class model
- ▶ 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" on page 344
- ▶ "Permissions" on page 344

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 347.

Permissions

The administrator provides login credentials for connecting with the API. The API client needs the username and password of an integration user defined in the UCMDB. These users do not represent human users of UCMDB, but rather applications that connect to UCMDB.

For details, see "Create an Integration User" on page 346.

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 UCMDb 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 = 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>getImpactAnalysisService</code>	Analyzing the effect of a change in the IT universe
<code>getTopologyQueryService</code>	Getting information about the IT universe
<code>getTopologyUpdateService</code>	Changing information in the IT universe

The client communicates with the server over HTTP.

Tasks

Put the API Jar File in the Classpath

The use of this API set requires the file **cmdb-api.jar**. The file can be accessed after unzipping the UCMDB version 9.00 download package. The jar is located in the UCMDB_Java_API subfolder, which is alongside the version 9.00 setup.exe file. The jar is not installed during the version 9.00 installation process.

Put the jar file in the classpath before compiling or running your application.

Create an Integration User

Applications written with this API set must log on with integration user credentials.

Caution: Only an integration user can connect to the UCMDB through this API set. A connection attempt by other types of users may cause errors, even when using LDAP verification.

To create an integration user:

1 Launch the Web browser and enter the server address, as follows.

- ▶ For HP Universal CMDB: <http://localhost:8080/jmx-console>.
- ▶ For Business Service Management: http://localhost.<domain_name>:8080/jmx-console.

You may have to log in with a user name and password (the defaults are **admin/admin**).

2 Under UCMDB, click **service=UCMDB Security Services** to open the JMX MBEAN View page.

- 3** Locate the `java.lang.String createIntegrationUser()` operation.
 - Fill in the `userName` and password fields.
 - Click **Invoke**.
 - Either click **Back to MBean View** to create more users, or close the JMX console.
- 4** Log on to UCMDB as an administrator.
- 5** From the **Settings** tab, run **Package Manager**.
- 6** Click the **New** icon.
- 7** Enter a name for the new package, and click **Next**.
- 8** In the Resource Selection tab, under Settings, click **Integration Users**.
- 9** Select a user or users that you created using the JMX console.
- 10** Click **Next** and then **Finish**. Your new package appears in the Package Name list in Package Manager.
- 11** Deploy the package to the users who will run the API applications.

For details, see "Deploy a Package" in the *HP UCMDB Administration Guide*.

Reference

HP Universal CMDB API Reference

For full documentation on the available APIs, see Chapter 10, "Introduction to APIs."

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 UCMDB" on page 348
- "Querying the UCMDB" on page 348
- "Querying the Class Model" on page 349
- "Analyzing Change Impact" on page 349

Populating the UCMDB

Use cases:

- A third-party asset management updates the UCMDB with information available only in asset management
- A number of third-party systems populate the UCMDB 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 UCMDB query capabilities

Querying the UCMDB

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 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 UCMDB. 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 UCMDB 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:

- ▶ "Entry Point Example" on page 350
- ▶ "Query Examples" on page 350
- ▶ "Topology Query Example" on page 352
- ▶ "Topology Update Example" on page 353
- ▶ "Impact Analysis Example" on page 353

Entry Point Example

```
final String HOST_NAME = "localhost";
final int PORT = 8080;
UcmdbServiceProvider provider =
    UcmdbServiceFactory.getServiceProvider(HOST_NAME, PORT);
final String USERNAME = "integration_user";
final String PASSWORD = "integration_password";
Credentials credentials =
    provider.createCredentials(USERNAME, PASSWORD),
ClientContext clientContext = provider.createClientContext("Example");
UcmdbService ucmdbService = provider.connect(credentials, clientContext);
```

Query Examples

The following examples demonstrate getting a single class definition and getting a list of all CIT definitions and their attributes.

Retrieving a Class Definition

```
ClassModelService classModelService
    = ucmdbService.getClassModelService();
String typeName = "disk";
ClassDefinition def =
    classModelService.getClassDefinition(typeName);
System.out.println("Type " + typeName + " is derived from type "
    + def.getParentClassName());
System.out.println("Has " + def.getChildClasses().size() +
    " derived types");
System.out.println("Defined and inherited attributes:");
for (Attribute attr : def.getAllAttributes().values()) {
    System.out.println("Attribute " + attr.getName() +
        " of type " + attr.getType());
}
```

Retrieving the List of CIT Definitions and Attributes

This example queries the attributes for one CIT and prints their names and types.

```
ClassModelService classModelService =
    ucmdbService.getClassModelService();
for (ClassDefinition def : classModelService.getAllClasses()) {
    System.out.println("Type " + def.getName() +
        " (" + def.getDisplayName() + ") is derived from type "
        + def.getParentClassName());
    System.out.println
        ("Has " + def.getChildClasses().size() + " derived types");
    System.out.println
        ("Defined and inherited attributes:");
    for (Attribute attr : def.getAllAttributes().values()) {
        System.out.println
            ("Attribute " + attr.getName() +
                " of type " + attr.getType());
    }
}
```

 **Topology Query Example**

```

TopologyQueryService queryService =
    ucmdbService.getTopologyQueryService();
TopologyQueryFactory queryFactory =
    queryService.getFactory();
QueryDefinition queryDefinition =
    queryFactory.createQueryDefinition
        ("Get hosts with more than one network interface");
String hostNodeName = "Host";
QueryNode hostNode =

queryDefinition.addNode(hostNodeName).ofType("host").queryProperty("display_label"
);
QueryNode ipNode =
    queryDefinition.addNode("IP").ofType("ip").queryProperty("ip_address");
hostNode.linkedTo(ipNode).withLinkOfType("contained").atLeast(2);
Topology topology = queryService.executeQuery(queryDefinition);
Collection<TopologyCI> hosts = topology.getCIsByName(hostNodeName);
for (TopologyCI host : hosts) {
    System.out.println("Host " + host.getPropertyValue("display_label"));
    for (TopologyRelation relation : host.getOutgoingRelations()) {
        System.out.println
            (" has IP " + relation.getEnd2CI().getPropertyValue("ip_address"));
    }
}
}

```


Topology Update Example

```
TopologyUpdateService topologyUpdateService =
    ucmdbService.getTopologyUpdateService();
TopologyUpdateFactory topologyUpdateFactory =
    topologyUpdateService.getFactory();
TopologyModificationData topologyModificationData =
    topologyUpdateFactory.createTopologyModificationData();
CI host = topologyModificationData.addCI("host");
host.setPropertyValue("host_key", "test1");
CI ip = topologyModificationData.addCI("ip");
ip.setPropertyValue("ip_address", "127.0.0.10");
ip.setPropertyValue("ip_domain", "DefaultDomain");
topologyModificationData.addRelation("contained", host, ip);
topologyUpdateService.create
    (topologyModificationData, CreateMode.IGNORE_EXISTING);
```

Impact Analysis Example

```
ImpactAnalysisService impactAnalysisService =
    ucmdbService.getImpactAnalysisService();
ImpactAnalysisFactory impactFactory =
    impactAnalysisService.getFactory();
ImpactAnalysisDefinition definition =
    impactFactory.createImpactAnalysisDefinition();
definition.addTriggerCI(disk).withSeverity
    (impactFactory.getSeverityByName("Warning(2)"));
definition.useAllRules();
ImpactAnalysisResult impactResult =
    impactAnalysisService.analyze(definition);
AffectedTopology affectedCIs =
    impactResult.getAffectedCIs();
for (AffectedCI affectedCI : affectedCIs.getAllCIs()) {
    System.out.println("Affected " +
        affectedCI.getType() + " " + affectedCI.getId() +
        " - severity " + affectedCI.getSeverity());
}
```


13

Data Flow Management Web Service API

This chapter includes:

Concepts

- ▶ Data Flow Management Web Service API Overview on page 355
- ▶ Data Flow Management Web Service API on page 356

Reference

- ▶ Data Flow Management Methods on page 357
- ▶ Data Flow Management Adding Credentials Example on page 360

Concepts

Data Flow Management Web Service API Overview

This chapter explains how third-party or custom tools can use the Web Service to manage discovery and integration.

For full documentation on the available operations, see *HP Discovery and Dependency Mapping Schema Reference*. These files are located in the following folder:

- ▶ **For HP Universal CMDB:** C:\hp\UCMDB\UCMDBServer\j2f\AppServer\webapps\site.war\amdocs\eng\doc_lib\DevRef_guide\DDM_Schema\webframe.html
- ▶ **For Business Service Management:** \\<Business Service Management Gateway Server root directory>\AppServer\webapps\site.war\amdocs\eng\doc_lib\DevRef_guide\DDM_Schema\webframe.html

Data Flow Management Web Service API

DFM Web Service is an API used to integrate applications with HP Universal CMDB (UCMDB). 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 on domains and Probes

Users of the Data Flow Management 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

Permissions

The administrator provides login credentials for connecting with the Web service. The required credentials depend on whether you are using Data Flow Management with a standalone version of HP Universal CMDB or from within Business Service Management.

When permissions are assigned through DFM, the permission levels are View, Update, and Execute. When they are assigned using Business Service Management, 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, see *Data Flow Management Schema Reference*.

To assign permissions:

- **When running HP Universal CMDB.** Log in using the credentials of a DFM user who has been granted permissions on the discovery resources. For details, see "Assign Access Rights" in the *HP UCMDB Administration Guide*.
- **When running Business Service Management.** Log in using the credentials of a Business Service Management user. The user must have been granted the relevant permissions on the ODB Web Service resource in Business Service Management.

Reference

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 *Data Flow Management Schema Reference*.

This section includes the following topics:

- "Managing DFM Job Methods" on page 357
- "Managing Trigger Methods" on page 358
- "Domain and Probe Data Methods" on page 358
- "Credentials Data Methods" on page 359
- "Data Refresh Methods" on page 359

Managing DFM Job Methods

- **activateJob**
Activates the specified job.
- **deactivateJob**
Deactivates the specified job.
- **dispatchAdHocJob**

Dispatches a job on the Probe ad-hoc. The job must be active and contain the specified trigger CI.

► **getDiscoveryJobsNames**

Returns the list of job names.

► **isJobActive**

Checks whether the job is active.

Managing Trigger Methods

► **addTriggerCI**

Adds a new trigger CI to the specified job.

► **addTriggerTQL**

Adds a new trigger TQL to the specified job.

► **disableTriggerTQL**

Prevents the TQL from triggering the job, but does not permanently remove it from the list of queries that trigger the job.

► **removeTriggerCI**

Removes the specified CI from the list of CIs that trigger the job.

► **removeTriggerTQL**

Removes the specified TQL from the list of queries that trigger the job.

► **setTriggerTQLProbesLimit**

Restrict the Probes in which the TQL is active in the job to the specified list.

Domain and Probe Data Methods

► **getDomainType**

Returns the domain type.

► **getDomainsNames**

Returns the names of the current domains.

► **getProbeIPs**

Returns the IP addresses of the specified Probe.

► **getProbesNames**

Returns the names of the Probes in the specified domain.

► **getProbeScope**

Returns the scope definition of the specified Probe.

► **isProbeConnected**

Checks whether the specified Probe is connected.

► **updateProbeScope**

Sets the scope of the specified Probe, overriding the existing scope.

Credentials Data Methods

► **addCredentialsEntry**

Adds a credentials entry to the specified protocol for the specified domain.

► **getCredentialsEntriesIDs**

Returns the IDs of the credentials defined for the specified protocol.

► **getCredentialsEntry**

Returns the credentials defined for the specified protocol. Encrypted attributes are returned empty.

► **removeCredentialsEntry**

Removes the specified credentials from the protocol.

► **updateCredentialsEntry**

Sets new values for properties of the specified credentials entry.

Data Refresh Methods

► **rediscoverCIs**

Locates the triggers that discovered the specified CI objects and reruns those triggers. (Note that the rerun command takes higher priority than other scheduled items.)

rediscoverCIs runs asynchronously. Call **checkDiscoveryProgress** to determine when the rediscovery is complete.

► **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.

► **rediscoverViewCIs**

Locates the triggers that created the data to populate the specified view, and reruns those triggers. (Note that the rerun command takes higher priority than other scheduled items.)

rediscoverViewCIs runs asynchronously. Call **checkViewDiscoveryProgress** to determine when the rediscovery is complete.

► **checkViewDiscoveryProgress**

Returns the progress of the most recent **rediscoverViewCIs** call on the specified view. The response is a value from 0-1. When the response is 1, the **rediscoverCIs** call has completed.

Data Flow Management Adding Credentials Example

This section contains an example of adding credentials to the Web service. For full documentation of the request and response for each operation, see *Data Flow Management Schema Reference*.

This section includes the following topics:

- "Adding Credentials Example" on page 361

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 = 8080;

    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 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 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
            IsProbeConnectedResponce 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.AUTHENTIC
ATE,auth);

        discoveryService = serviceStub;
    } catch (Exception e) {
        throw new Exception("cannot create a connection to service ", e);
    }

    return discoveryService;
}
} // End class
```

Index

A

- accessing data
 - guidelines 28
- adapter
 - add for new external data source 134
 - deploying 139, 172, 173
 - loading 173, 174
- adapter capabilities 148
- adapter.conf 180
- adapters
 - assigning jobs to 43
 - configuration file for Service Manager 228
 - configuration file in ServiceCenter/Service Manager 220
 - creating 33
 - defining input (Trigger CIT, Input TQL) 34
 - defining output 40
 - deployment for ServiceCenter/Service Manager 228
 - deployment of ServiceDesk adapter 229
 - development and testing 23
 - finding correct credentials for connections 56
 - implementing 32
 - interaction with the federation framework 118
 - interfaces 133
 - modifying existing 26
 - overriding parameters 41
 - packaging and productization 24
 - preparing package 162
 - prerequisites 157
 - scheduling 44
 - separating 30

- Trigger TQL 43
- usage in ServiceCenter/Service Manager 218
- validating setup 157
- writing new pattern 27

- adapter-writing
 - introduction 18
 - research stage 26

- API
 - Data Flow Management Web Service 355
 - UCMDB Java
 - UCMDB Java API 343
 - UCMDB Web service 267
- APIs
 - included with HP Universal CMDB 265
 - introduction 265
 - Web Service, overview 355

B

- blueprint 19
- Books Online 12

C

- character set
 - determining encoding 92
- configuration file
 - for Service Manager adapter 228
 - for ServiceCenter/Service Manager adapter 220
- configuration files for the generic database adapter 179
- configuration type
 - UCMDB Web service API 338
- content development and adapter-writing 17

Index

- converters
 - generic database adapter 196

D

Data Flow Management

- Web Service 355
- Web Service API 356
- Web Service APIs, overview 355
- Web Service, adding credentials
 - example 361
- Web Service, managing query
 - methods 357
- Web Service, mapping methods 357
- Web service, permissions 356

data source

- add adapter for new data source 134

database adapter

- configuration examples 201

deploying an adapter 139

derived properties 277

DFM

- adapters and related components 20
- code 61
- development cycle 21
- integration 24

DFM code

- recording 59

Discovery Analyzer 81

- troubleshooting and limitations 89
- working with 83

DiscoveryMain function 49

discriminator.properties 194

documentation updates 14

documentation, online 12

E

Eclipse

- configure to run Jython scripts in debug mode 71
- mapping between CI attributes and database tables 243

encoding

- determining for character sets 92

error messages 105

- conventions 107

- overview 106

- severity levels 110

- troubleshooting and limitations 111

- writing 107

- writing for specific locale 99

executeCommandAndDecode

- method 101

F

federated CMDB

- federation framework flow for FTQL 119

- federation framework flow for replication 131

federated database adapter

- supported TQL queries 153

- troubleshooting 214

federation

- adapter capabilities 148

federation framework

- adapter and mapping interaction 118

- adapter interfaces 133

- overview 114

federation framework SDK 113

fixed_values.txt 196

Framework instance 53

G

generic database adapter

- converters 196

- federated database configuration files 179

- overview 153

- plugins 200

- reconciliation 154

getCharsetName

- method 102

getLanguageBundle

- method 102

H

Hibernate mapping tool 155

HP Data Flow Management API Reference 32

HP Software Support Web site 13

HP Software Web site 14

J

Java

UCMDB API 343

Java exceptions

handling 59

Jython

configure Eclipse to run in debug mode 71

generating results 51

libraries and utilities 64

structure of the file 48

using external Java JAR files 31

K

Knowledge Base 13

L

log files

enabling 179

for federated database 211

logger.py 65

logs

severity levels 110

M

mapping

interaction with the federation framework 118

methods

executeCommandAndDecode 101

getCharsetName 102

getLanguageBundle 102

useCharset 102

modeling.py 67

multi-lingual locales

adding support 91

adding support for new language 95

API reference 100

changing default 99

decoding commands without keyword 98

overview 91

writing a new job 97

multi-threading 228

N

netutils.py 67

O

online documentation 12

Online Help 13

online resources 13

orm.xml 183

osLanguage 102

P

persistence.xml 193

plugins

generic database adapter 200

properties

derived 277

Q

queries

UCMDB Web service API 270

R

Readme 12

reconciliation_rules.txt 189

reconciliation_types.txt 189

relation

UCMDB Web service API 339

replication_config.txt 196

reports

viewing 178

resource bundles 94

S

scripts

modifying out of the box 46

ServiceCenter/Service Manager adapter deployment 228

Index

- add attribute to CIT 233
- ServiceDesk adapter
 - deployment 229
- shellutils.py 67
- simplifiedConfiguration.xml 181
- SSL communication 241

T

- TopologyMap
 - UCMDB Web service API 270
- TQL
 - supported queries in federated database adapter 153
- transformations.txt 192
- Troubleshooting and Knowledge Base 13

U

- UCMDB
 - querying
 - Web service 274
- UCMDB Java API
 - application structure 345
 - integration user, creating 346
 - jar file 346
 - permissions 344
 - using 344
- UCMDB Web service API
 - using 268
- UCMDB Web service API
 - addCIsAndRelations 298
 - addCustomer 299
 - calculateImpact 301
 - chunkInfo 341
 - CIT name 338
 - class name 338
 - configuration type name 338
 - deleteCIsAndRelations 300
 - errors 274
 - exceptions 274
 - executeTopologyQueryByName 284
 - executeTopologyQueryByNameWith Parameters 284
 - executeTopologyQueryWithParameters 285
 - getAllClassesHierarchy 282

- getChangedCIs 286
- getCIsByID 288
- getCIsByType 289
- getClassAncestors 281
- getCmdbClassDefinition 282
- getFilteredCIsByType 290
- getImpactPath 302
- getImpactRulesByNamePrefix 303
- getQueryNameOfView 294
- getTopologyQueryExistingResultByName 295
- getTopologyQueryResultCountByName 295
- identifier in impact analysis methods 283
- inherited properties query 293
- key attributes 337
- labels 270
- parameter format 279, 336, 340
- permissions 269
- query methods 283
- query the UCMDB class model 281
- query, properties returned 276
- relation 339
- removeCustomer 300
- ShallowRelation 340
- TopologyMap 270
- TQL queries 270
- update methods 298, 301
- updateCIsAndRelations 300
- Web Service, calling 274

- updates, documentation 14
- useCharset
 - method 102

V

- views
 - creating 174, 176, 177

W

- Web Service
 - Data Flow Management 355
 - UCMDB API 267
 - UCMDB Web service API 274
- Web Service APIs

overview 355
What's New 12

